Predictable and Secure Computing Infrastructure for Intelligent Cyber-Physical Systems

2/14/2020

Heechul Yun

Associate Professor, EECS

University of Kansas

KU THE UNIVERSITY OF KANSAS

Intelligent Cyber-Physical Systems

- Cyber Physical Systems (CPS)
  - Cyber (Computer) + Physical (Plant)
- Real-time
  - Control physical process in real-time
- Safety-critical
  - Can harm people/things
- Intelligent
  - Can function autonomously

Cost, Size, Weight, and Power Constraints

- Maximum performance with minimal resources
  - Cannot afford too many or too power hungry systems

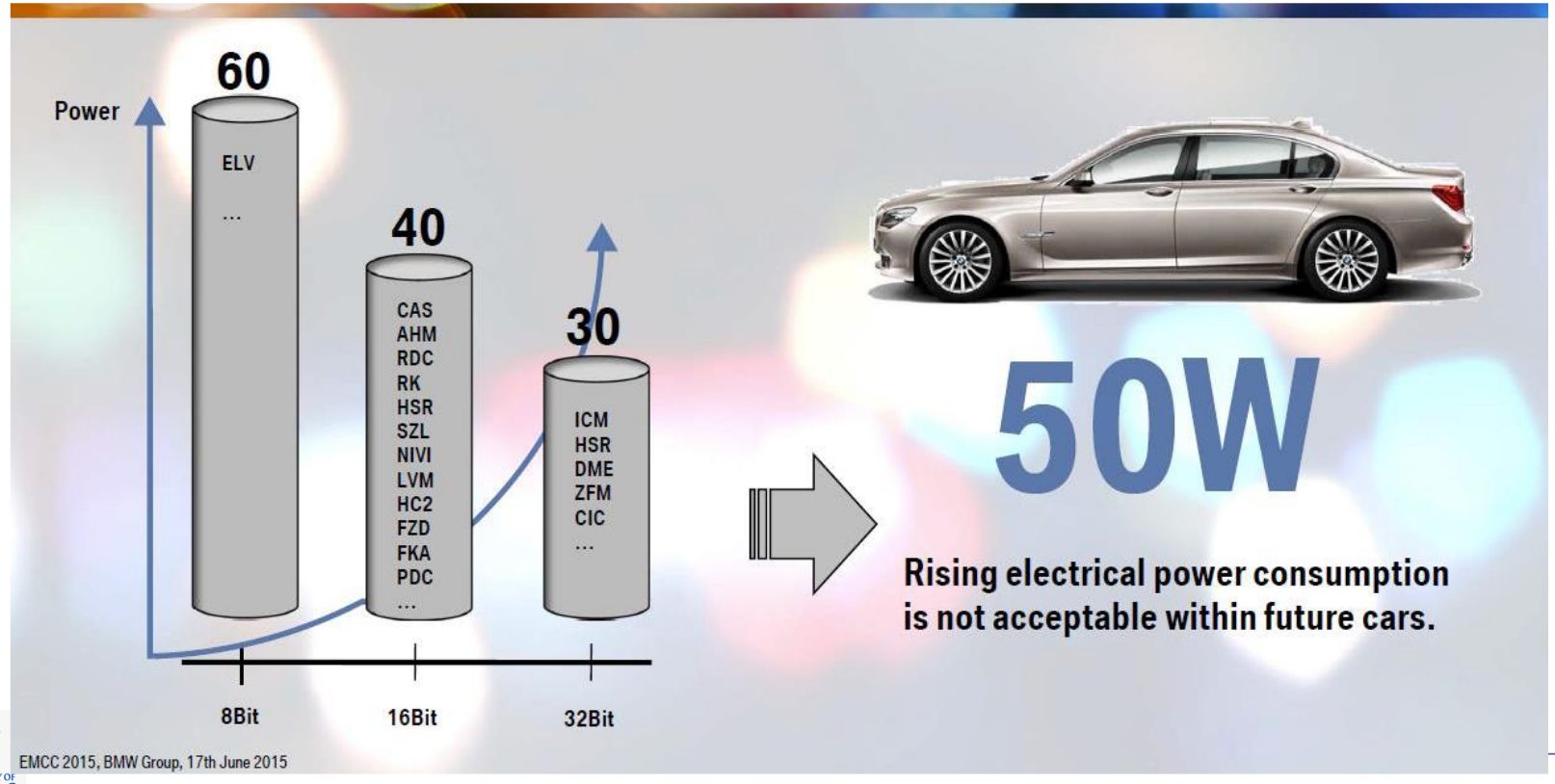
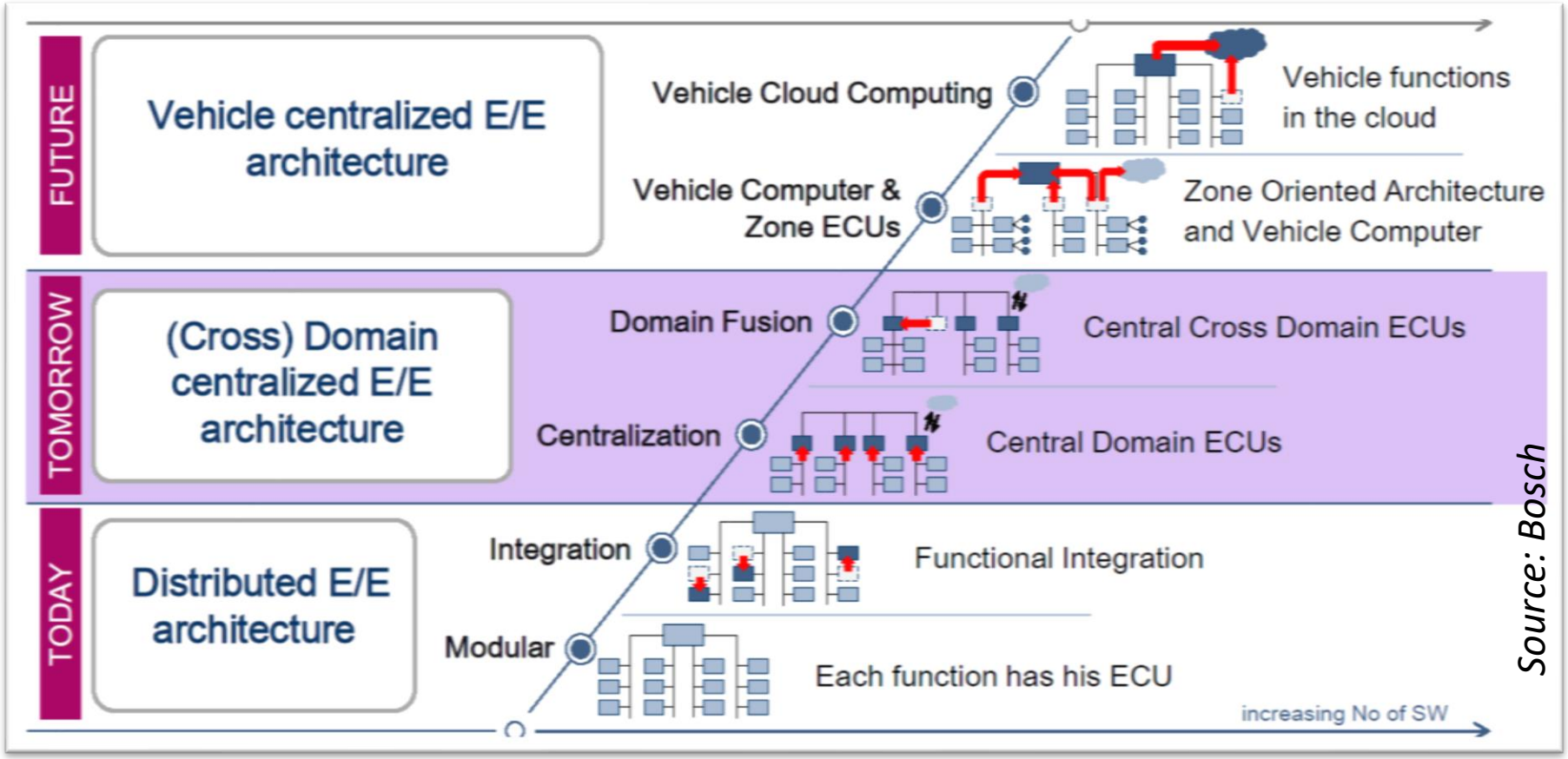


Figure source: Robert Leibinger, "Software Architectures for Advanced Driver Assistance Systems (ADAS)," OSPERT, 2015

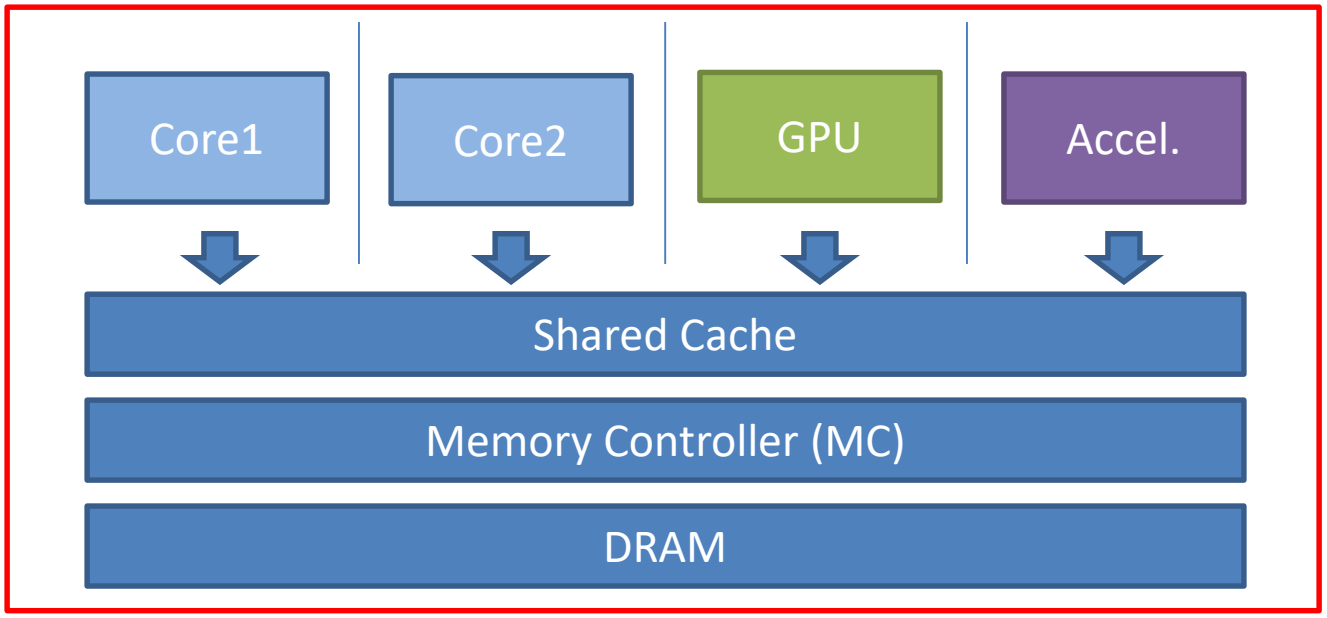
Trends in Automotive E/E Systems

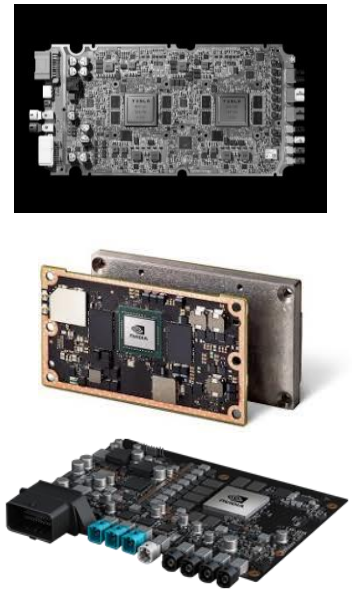


A. Hamann. "Industrial Challenge: Moving from Classical to High-Performance Real-Time Systems." WATER, 2018.

Centralization & High-Performance HW

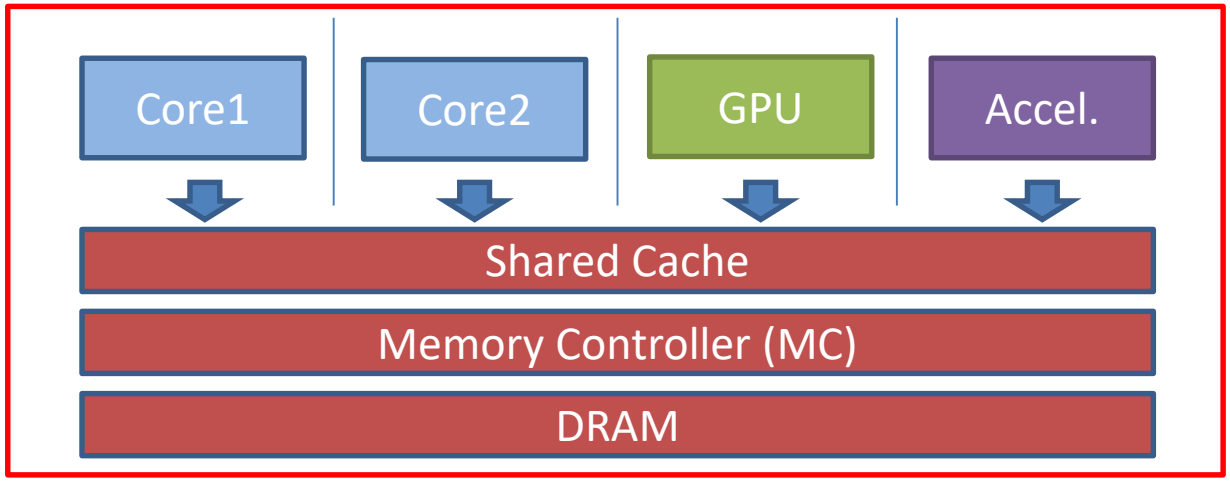
Modern System-on-Chip (SoC)





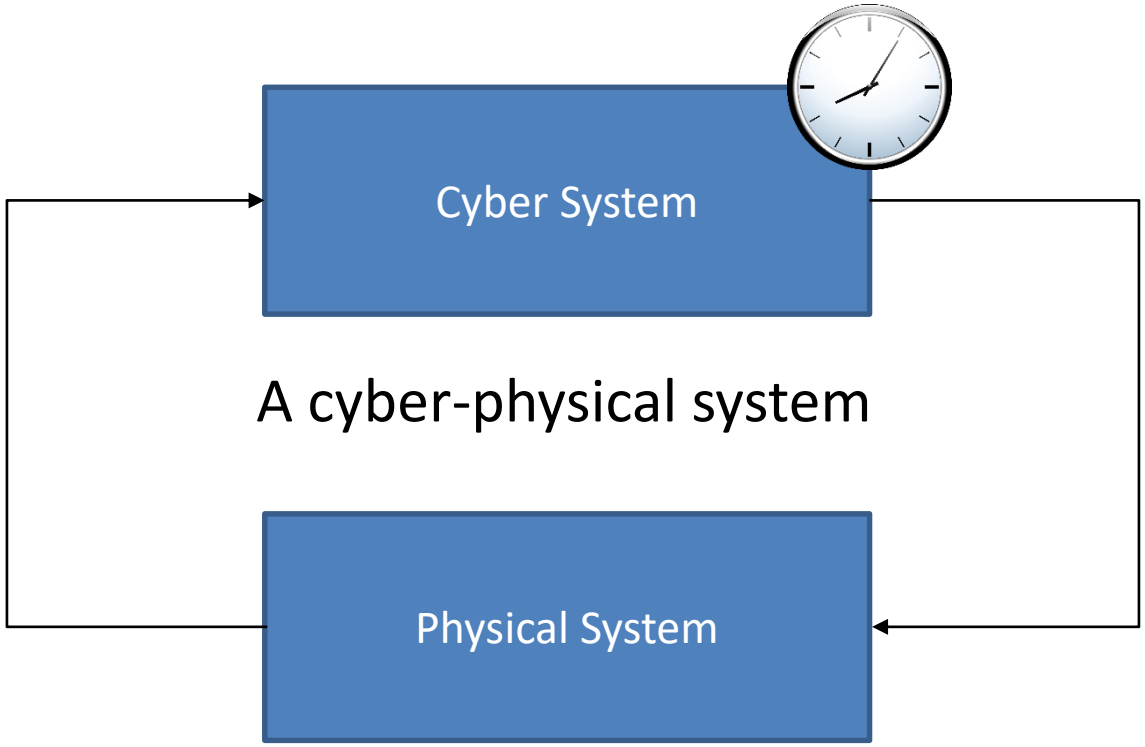
- Integrate multiple cores, GPU, accelerators
- Good performance, cost, size, weight, power
- **But…**

Challenge: Time Predictability



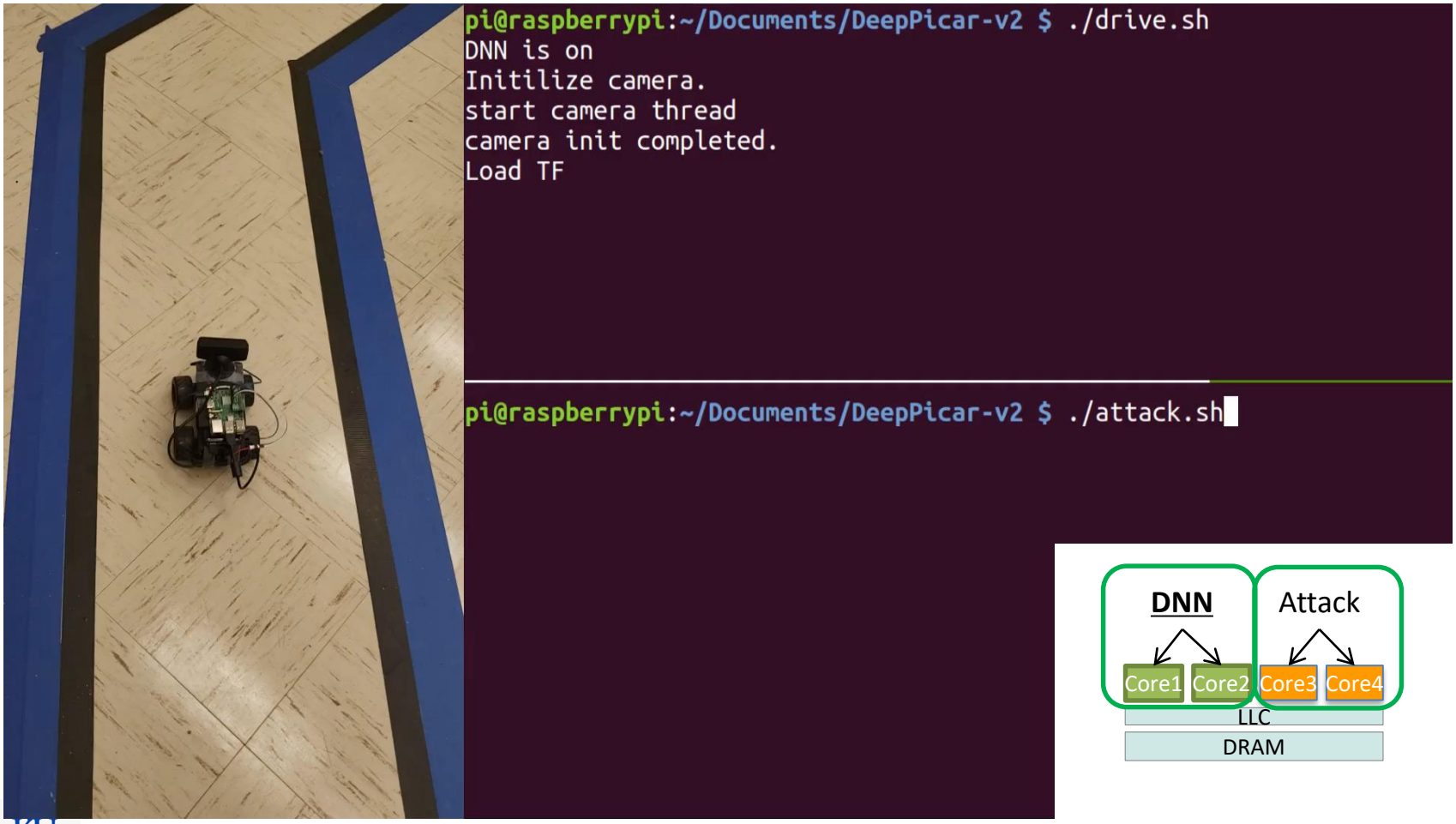
- Many important hardware resources are *shared*
- Each is locally optimized for average performance
- Software has limited ability to reason and control
- Unpredictable software execution timing

Why Predictable Timing?



- Computing time matters for temporal correctness
  - Logically correct output at wrong time is a fault

Effect of Co-Scheduling



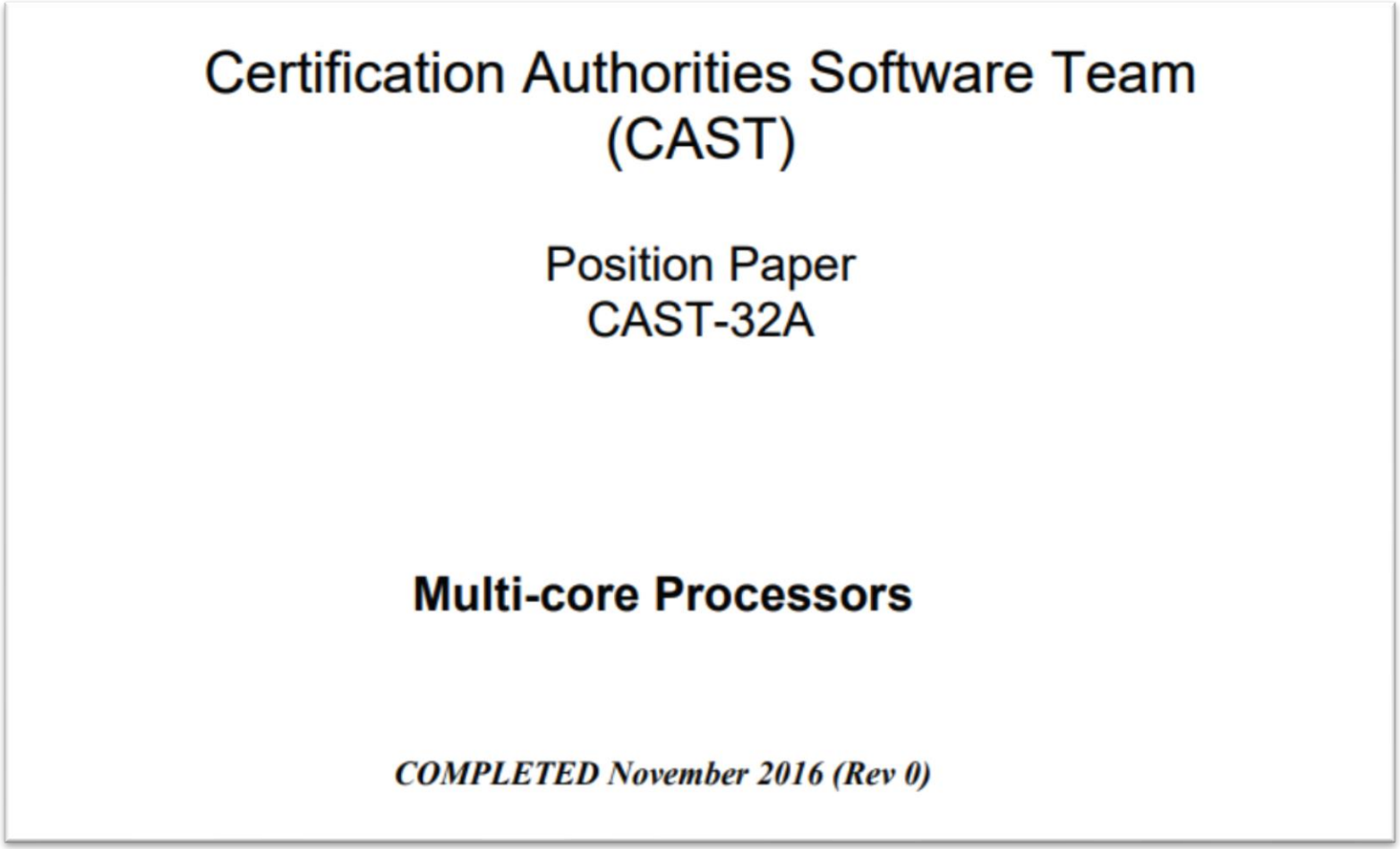
https://youtu.be/Jm6KSDqlqiU

Automotive Industry Challenges

- “Automotive industry is transitioning from µC toward µP based platforms”
- “Interference effects (on µPs) are more severe by orders of magnitude compared to µC platforms”
- “Goal for automotive systems engineering: predictable real-time behavior on high-performance platforms”

(*) Arne Hamann (Bosch), “Industrial Challenges: moving from classical to high-performance real-time systems." In Waters 2019

Certification Challenges in Aviation



Certification Authorities Software Team
(CAST)

Position Paper
CAST-32A

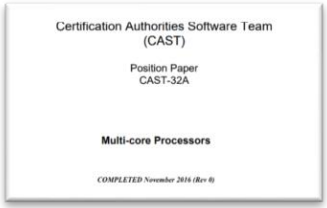
Multi-core Processors

COMPLETED November 2016 (Rev 0)

https://www.faa.gov/aircraft/air_cert/design_approvals/air_software/cast/cast_papers/media/cast-32A.pdf

KU THE UNIVERSITY OF KANSAS

CAST-32A: Multicore-Processors


Certification Authorities Software Team
(CAST)

Position Paper
CAST-32A

Multi-core Processors

COMPLETED November 2016 (Rev 0)

- A position paper by FAA and other certification agencies on multicore
- Discuss **interference channels** of multi-core that affect software timing
- Suggestion 1: disable all but one core
- Suggestion 2: provide evidence that all interference channels are taken care of ("robust partitioning") → **nobody can (yet)**

Timing Matters for Security

https://meltdownattack.com/

- Measurable timing differences in accessing shared hardware resources can leak secret

Fundamental Challenge: Isolation

- Traditionally about memory isolation
  - Prevent unauthorized access to memory
  - Hardware support: MPU, MMU
- What we need
  - **Prevent timing influence between domains**
  - Not only for real-time systems
  - But also for security[1]

KU THE UNIVERSITY OF KANSAS

[1] Q Ge, Y Yarom, T Chothia, G Heiser. "Time Protection: the Missing OS Abstraction". In *EuroSys*, 2019

My Research

- Build **predictable and secure computing infrastructure** for the next generation of intelligent Cyber Physical Systems (CPS).

Research Sponsors

Equipment Sponsors

KU THE UNIVERSITY OF KANSAS

Agenda

- **Part 1. Time predictable software on COTS hardware**

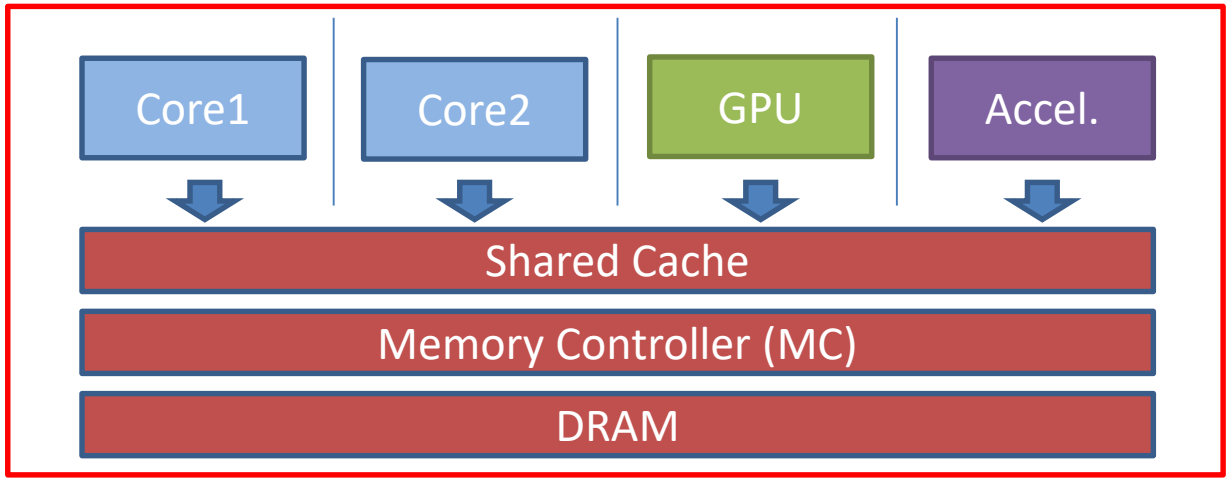
- Part 2. Hardware/software collaboration for time predictability and security

What are the (hidden) sources of interference of COTS hardware?

[**RTAS'16**] Prathap Kumar Valsan, Heechul Yun, Farzad Farshchi. Taming Non-blocking Caches to Improve Isolation in Multicore Real-Time Systems. In *IEEE Intl. Conference on Real-Time and Embedded Technology and Applications Symposium*, 2016.

[**RTAS'19-1**] Michael Garrett Bechtel and Heechul Yun. Denial-of-Service Attacks on Shared Cache in Multicore: Analysis and Prevention. *IEEE Intl. Conference on Real-Time and Embedded Technology and Applications Symposium*, 2019

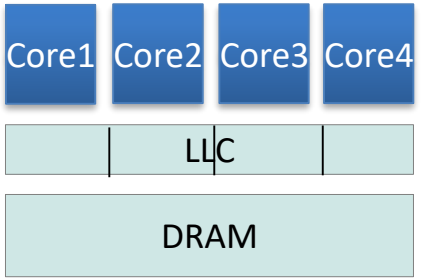
Shared Hardware Resources



- Lots of important hardware resources are shared
- Hardware makes allocation/scheduling decisions without knowing what software wants/does
- Existing partitioning techniques are not sufficient

Cache Partitioning

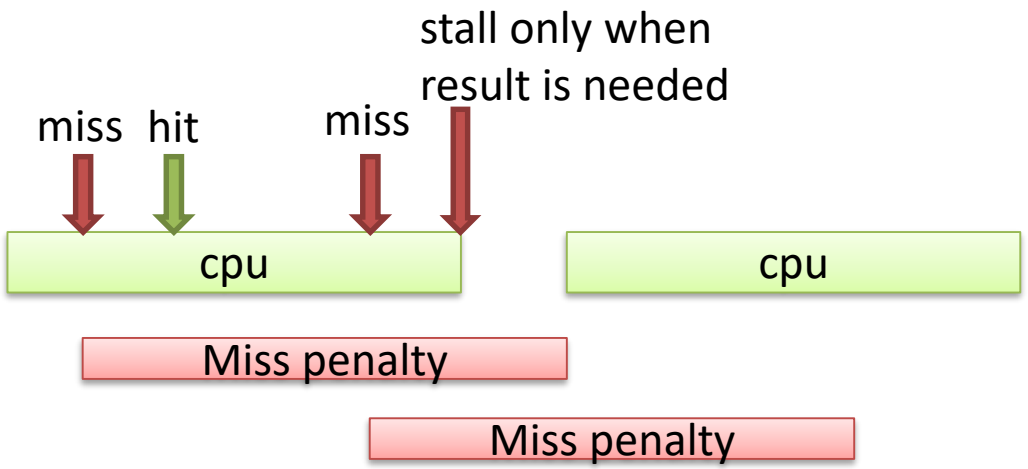
- Eliminate unwanted cache-line evictions
- Can be done in either SW or HW
  - Software: page coloring
  - Hardware: way partitioning



- Common assumption
  - Cache partitioning → performance isolation
- Not necessarily true for **non-blocking caches**

Non-Blocking Cache

Blocking cache



Non-blocking cache

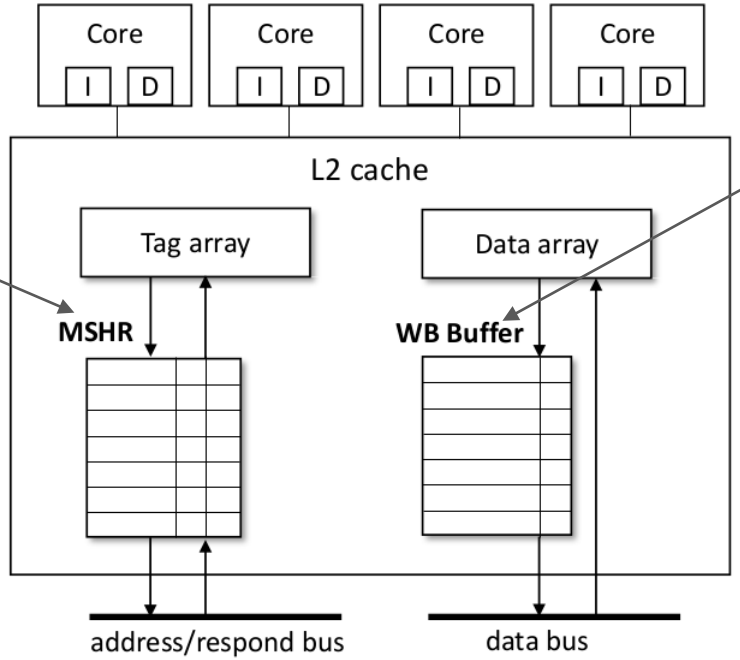
- Can serve cache hits under multiple cache misses
  - Essential for multicore performance

(*) D. Kroft. "Lockup-free instruction fetch/prefetch cache organization," ISCA'81

Non-Blocking Cache

Miss Status Holding Registers

- Track outstanding cache misses.



Writeback Buffer.

- Holds evicted dirty lines (writebacks).
- Prevents cache refills from waiting.

- Cache internal structures (MSHRs, writeback buffer) can have huge timing impacts if they are exhausted

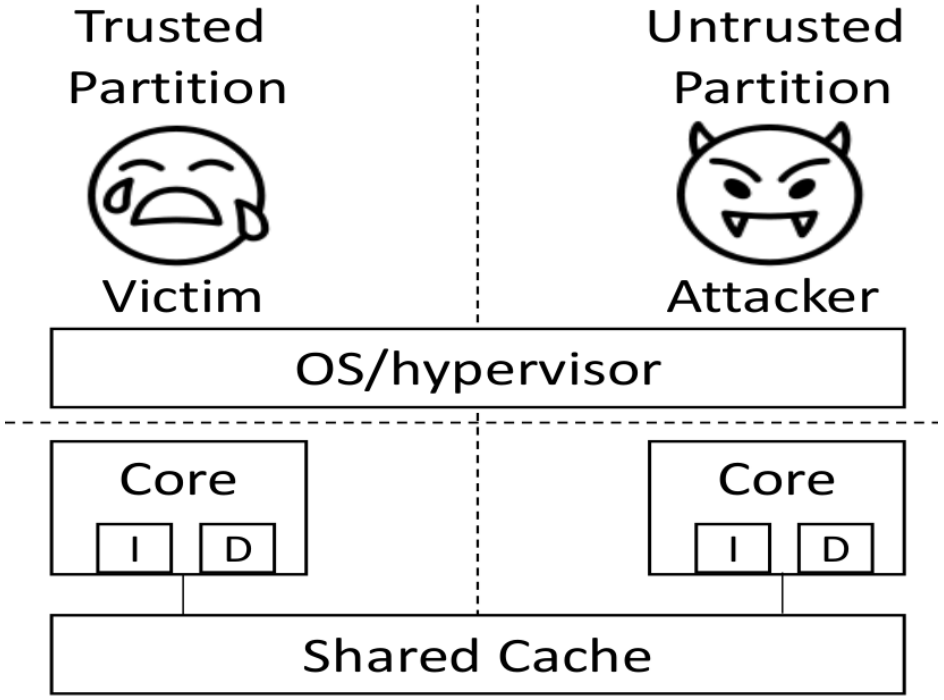
[**RTAS'16**] Prathap Kumar Valsan, Heechul Yun, Farzad Farshchi. Taming Non-blocking Caches to Improve Isolation in Multicore Real-Time S ystems. In *IEEE Intl. Conference on Real-Time and Embedded Technology and Applications Symposium*, 2016.

[**RTAS'19-1**] Michael Garrett Bechtel and Heechul Yun. Denial-of-Service Attacks on Shared Cache in Multicore: Analysis and Prevention. *IE EE Intl. Conference on Real-Time and Embedded Technology and Applications Symposium*, 2019

Cache Blocking

- Happens if MSHRs/WBBuffer become full
  - The cache is **locked up**
  - Subsequent accesses---including **cache hits**---to the cache **stall** and have to wait until the pending misses are completed
  - A pending miss costs **100's of CPU cycles** to complete (access to DRAM is slow)
  - We will see the impact of this in later experiments

Threat Model



- Attacker's goal: increase the victim's task **execution time**
- The attacker is on different core/memory/cache partition
- The attacker can only execute non-privileged code.

Cache DoS Attacks

```
for (i = 0; i < mem_size; i += LINE_SIZE)
{
        sum += ptr[i];
}
```

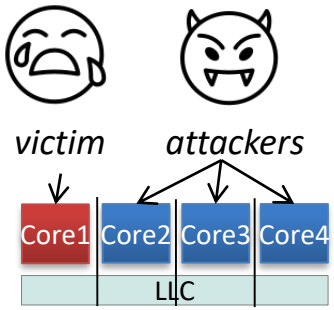
Read Attacker
(target **MSHRs**)

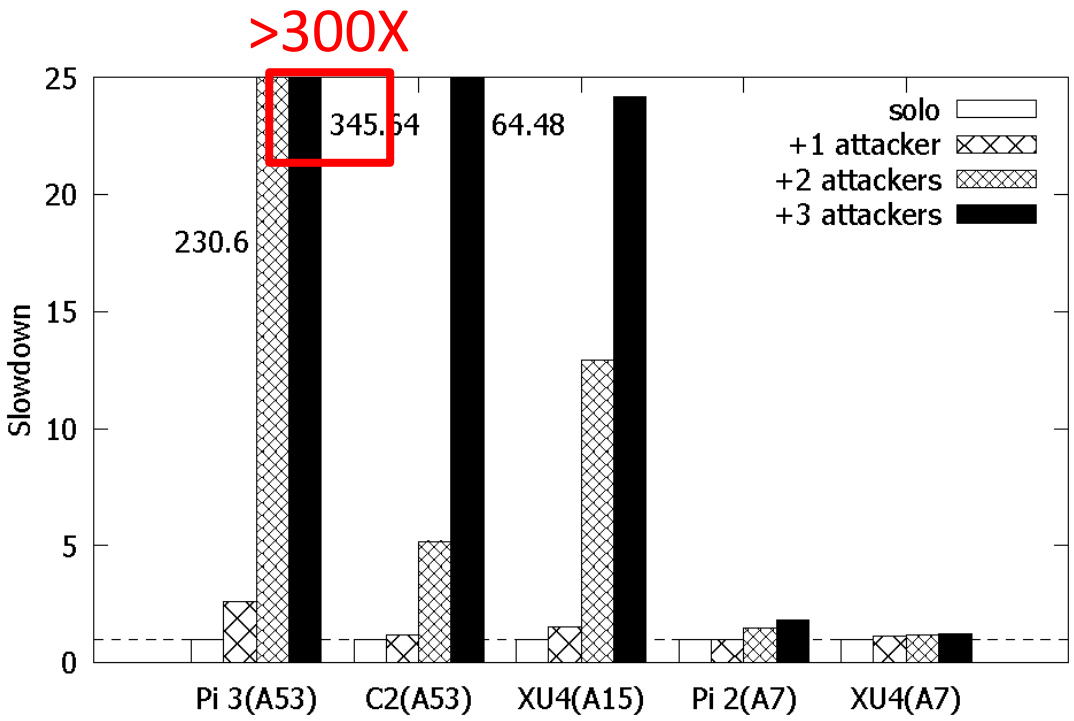
```
for (i = 0; i < mem_size; i += LINE_SIZE)
{
        ptr[i] = 0xff;
}
```

Write Attacker
(target **WBBuffer**)

- Denial-of-Service (DoS) attacks targeting internal hardware structures of a shared cache.
  - Block the cache → delay the victim's execution time

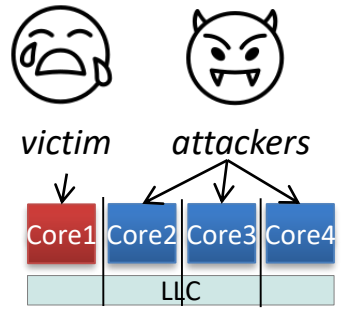
Effects of Cache DoS Attacks



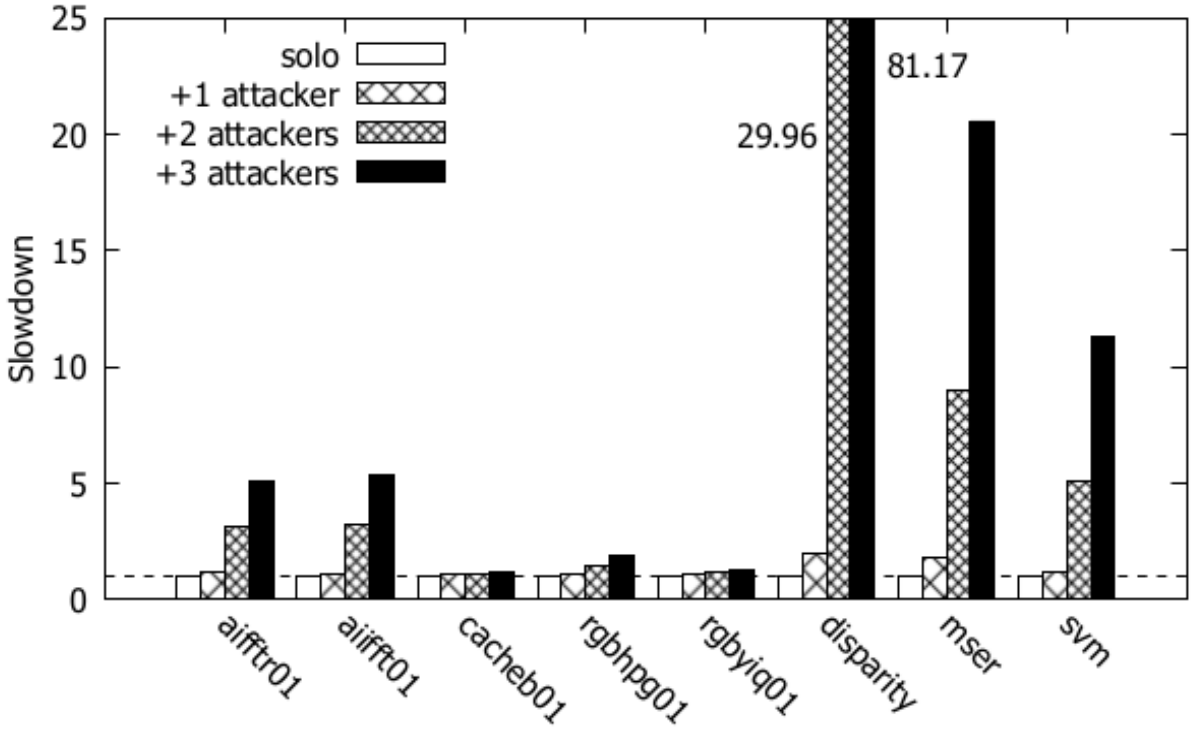


- Observed worst-case: >300X (times) slowdown
  - On popular in-order multicore processors
  - Due to contention in cache internal buffers

Effects on EEMBC and SD-VBS



Raspberry Pi 3 (A53)



- Cache DoS attacks are effective on real-world benchmarks
- LLC sensitive SD-VBS benchmarks are more susceptible

Summary

- Cache internal hardware structures (MSHRs, WriteBack buffer) are viable DoS attack vectors in multicore platforms.
- Traditional cache partitioning may not be effective to defend against cache DoS attacks targeting these internal hardware structures.

How to improve time predictability of software on COTS hardware?

[**RTAS'19-2**] Waqar Ali and Heechul Yun. RT-Gang: Real-Time Gang Scheduling Framework for Safety-Critical Systems. *IEEE Intl. Conference on Real-Time and Embedded Technology and Applications Symposium*, 2019

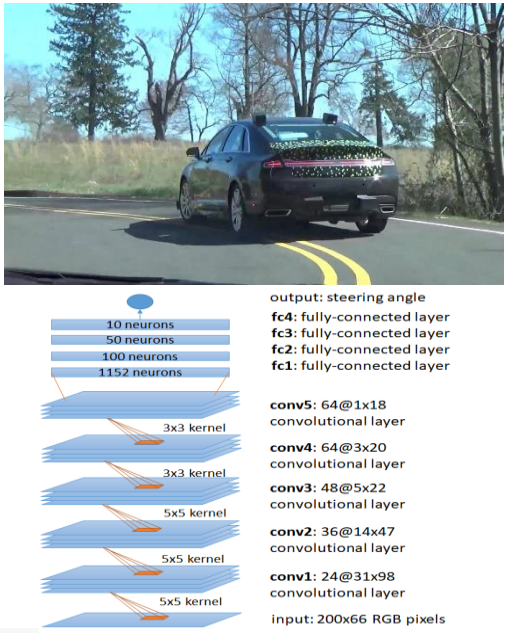
Waqar Ali, Rodolfo Pellizzoni, Heechul Yun. RT-Gang: Real-Time Gang Scheduling Framework for Safety-Critical Systems. 2020 (under submission)

KU THE UNIVERSITY OF KANSAS

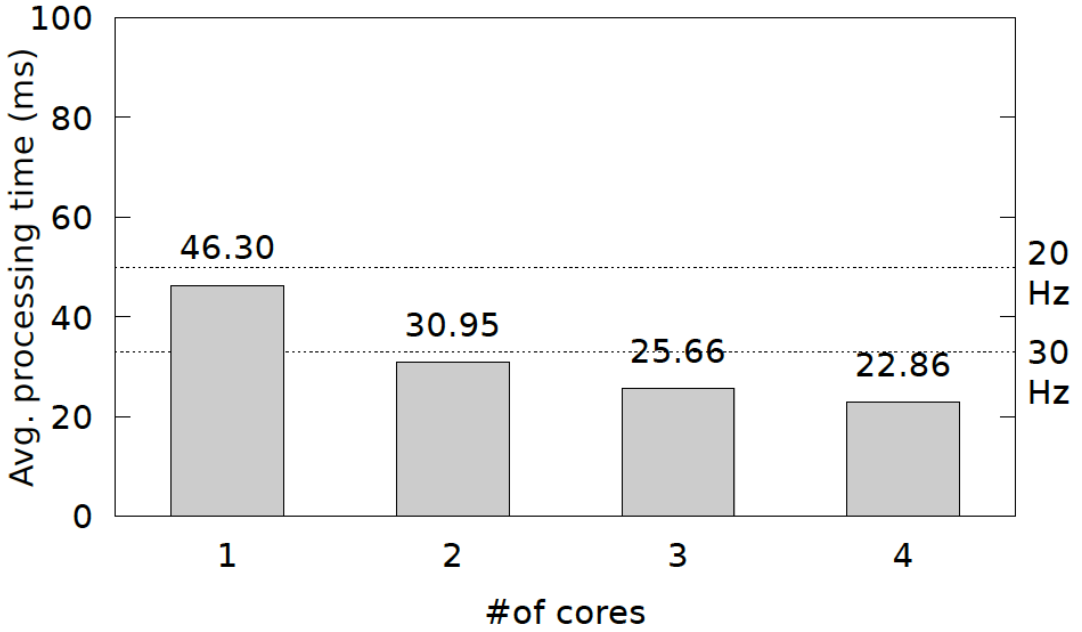
Parallel Real-Time Tasks

- Many emerging workloads in AI, vision, robotics are parallel real-time tasks

DNN based real-time control[+]



Effect of parallelization on DNN control task



[+] M. Bojarski, "End to End Learning for Self-Driving Cars." arXiv:1604.07316, 2016

Observations

- Constructive sharing (Good)
  - Between threads of a single parallel task
- Destructive sharing (Bad)
  - Between threads of different tasks

- **Goal: analyzable and efficient parallel real-time task scheduling framework for multicore**
  - By avoiding destructive sharing

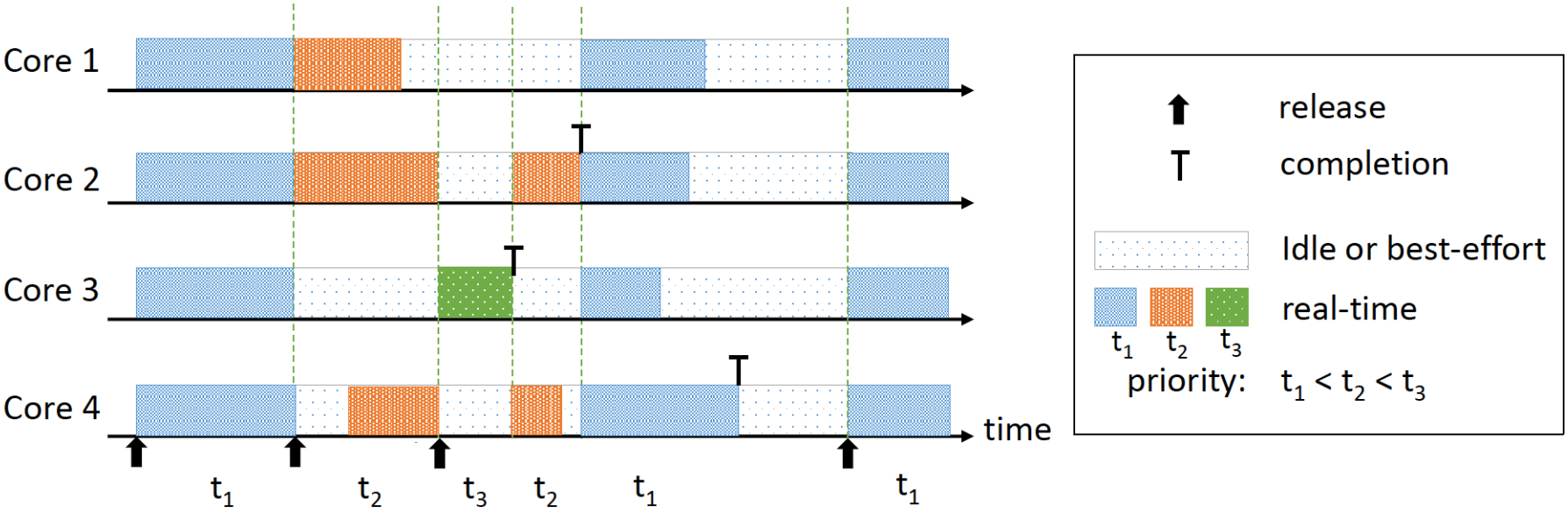
Gang Scheduling

- Schedule all threads of a parallel task *only if* enough cores are available

- Gang-FTP (fixed task priority)
  - Highest priority task τ_i on the first h_i available cores (if exist) among the active (ready) tasks.

$$\tau_i = < h_i , C_i , T_i >$$

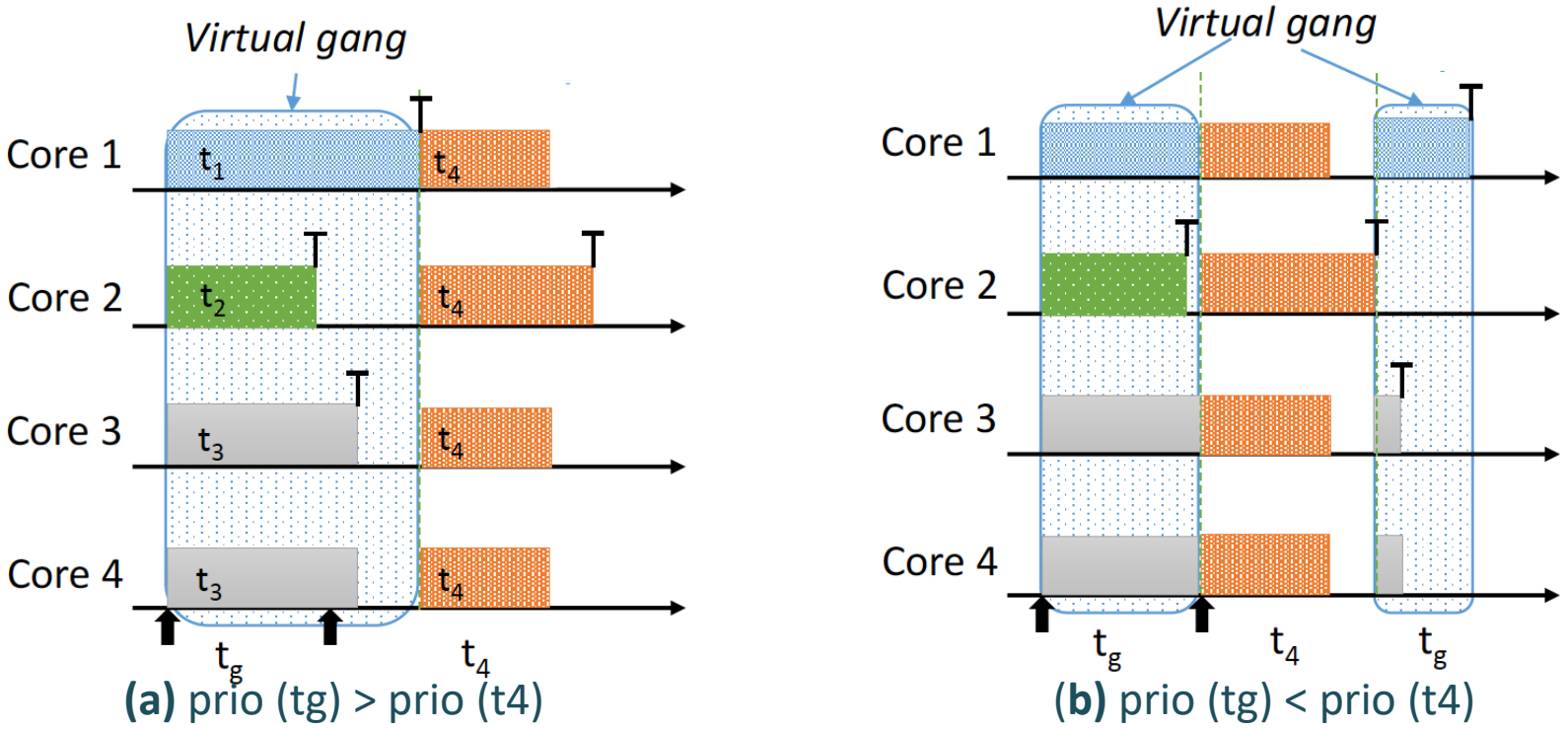
RT-Gang



- Restricted from of Gang-FTP: One gang at a time
  - Eliminate inter-task interference by construction
  - Allow best-effort co-scheduling with bandwidth limit
  - Cons: limited real-time task utilization

[**RTAS'19-2**] Waqar Ali and Heechul Yun. RT-Gang: Real-Time Gang Scheduling Framework for Safety-Critical Systems. *IEEE Intl. Conference on Real-Time and Embedded Technology and Applications Symposium*, 2019.

RTG-Sync



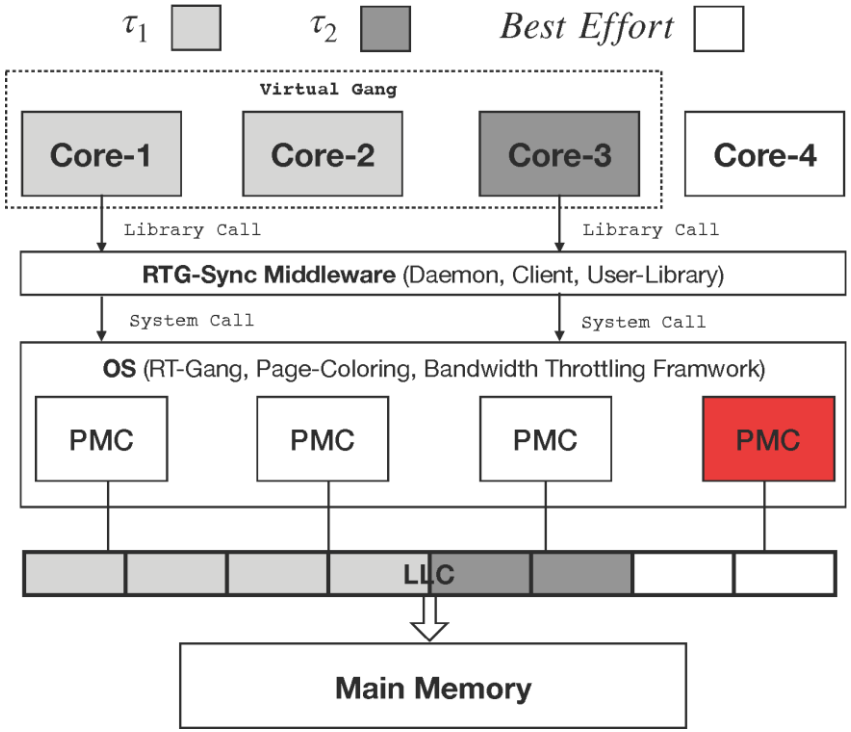
(a) prio (tg) > prio (t4)

(b) prio (tg) < prio (t4)

- Statically group RT tasks as a "**virtual gang**"
  - All members are *synchronously* released and scheduled
  - Improve real-time task schedulability

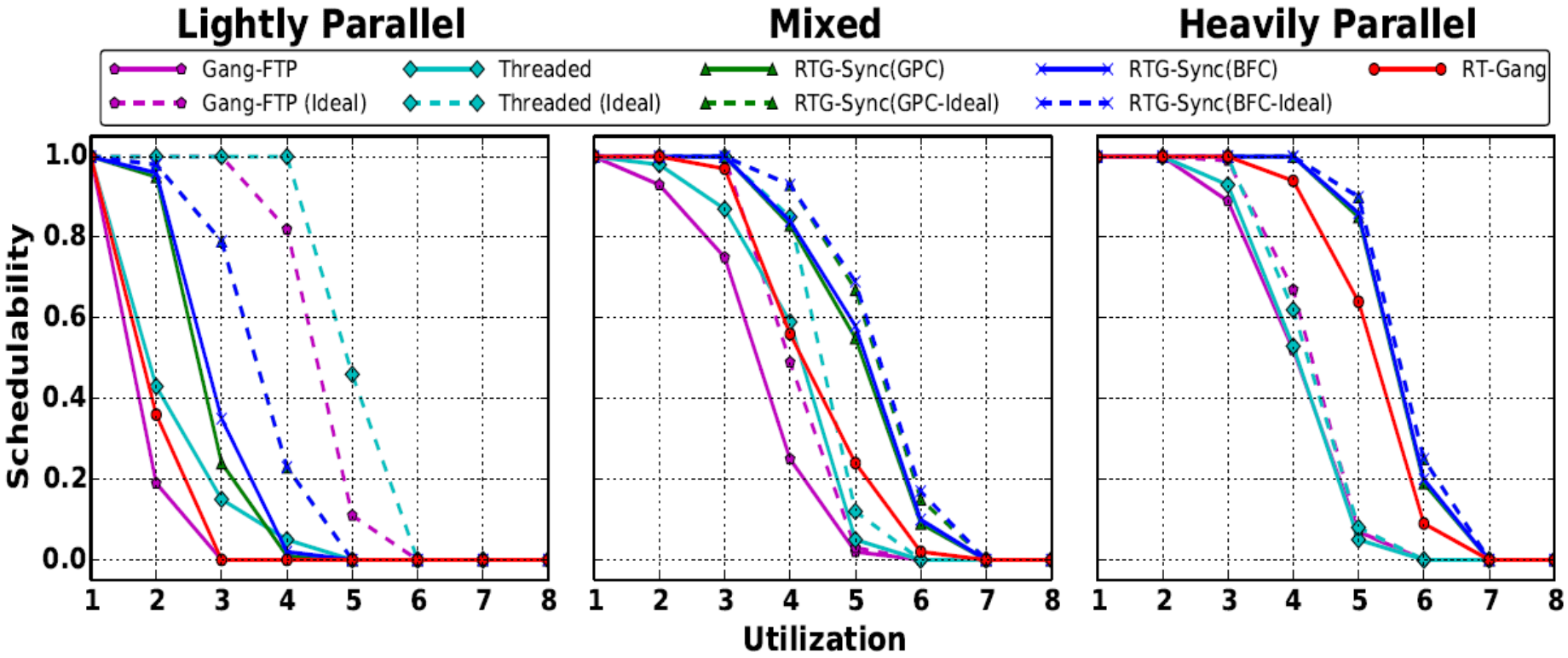
Waqar Ali, Rodolfo Pellizzoni, Heechul Yun. RT-Gang: Real-Time Gang Scheduling Framework for Safety-Critical Systems. 2020 (under submission)

RTG-Sync



- Gang scheduler + bandwidth limiter + cache partitioning, implemented in Linux kernel
- Middleware for virtual gang management

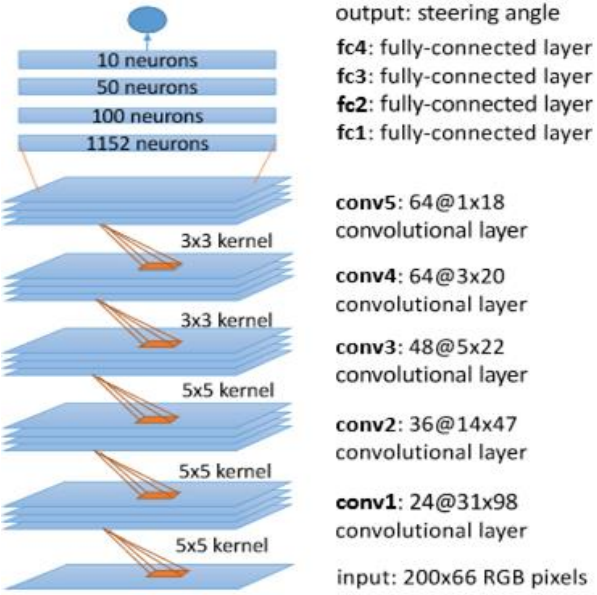
Schedulability Analysis

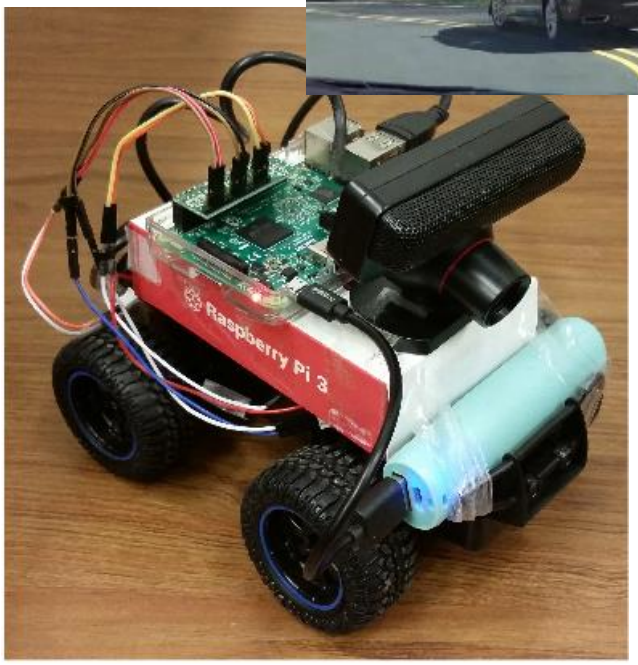


- RTG-Sync achieves better analytic schedulability
  - Compared to Gang-FTP, Threaded (Linux baseline)

Case Study: DeepPicar*

- A low cost, small scale replication of NVIDIA's DAVE-2
- **Uses the exact same DNN**
- Runs on a quad-core SoC in **real-time**





(*) Michael Garrett Bechtel, Elise McEllhiney, Minje Kim, Heechul Yun. "DeepPicar: A Low-cost Deep Neural Network-based Autonomous Car." In *RTCSA*, IEEE, 2018

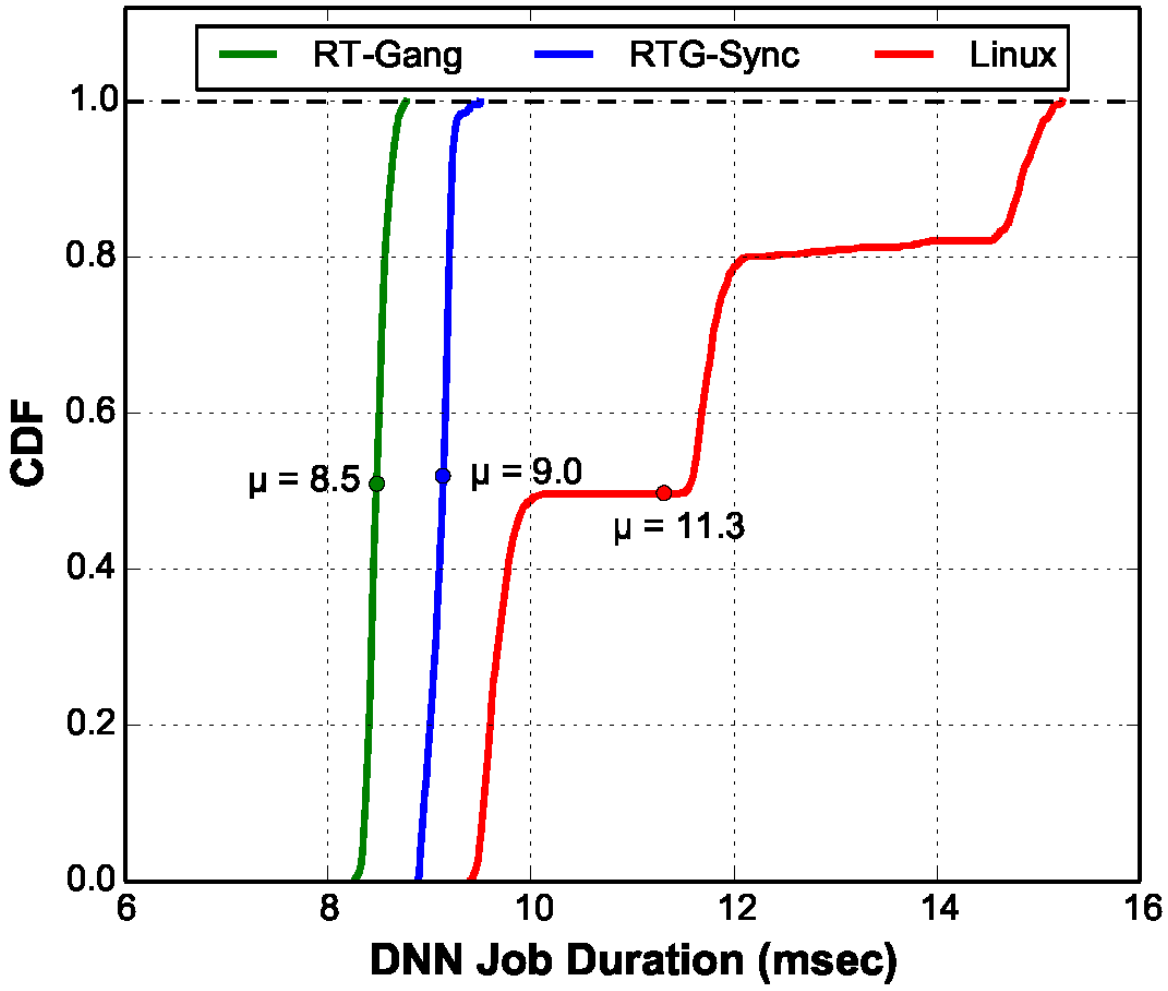
Experiment Setup

- NVIDIA Jetson TX2 (4 ARM cores)
- Parallel taskset
  - DeepPicar DNN control tasks, Parboil, IsolBench

Task	WCET (ms)	Period (ms)	# of Threads	Priority
τ^{RT}_{BWT}	50.0	100.0	4	5
τ^{RT}_{DNN-1}	8.2	50.0	2	10
τ^{RT}_{DNN-2}	8.2	50.0	2	10
τ^{BE}_{cutcp}	∞	N/A	2	N/A
τ^{BE}_{lbm}	∞	N/A	2	N/A

KU THE UNIVERSITY OF KANSAS

Execution Time Distribution

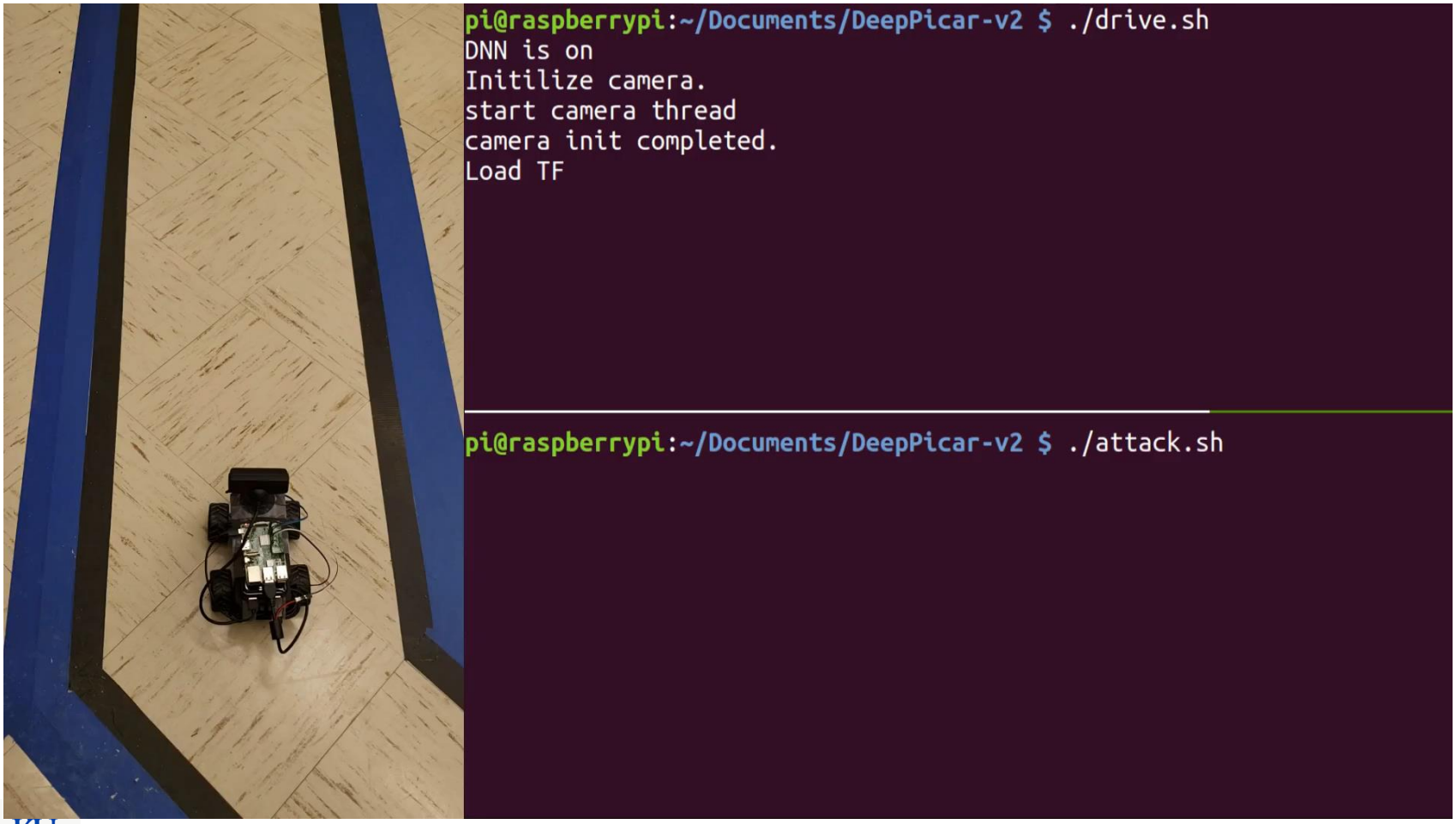


KU THE UNIVERSITY OF KANSAS

Execution Trace

Annotated outputs of *KernelShark* tool

Effect of RT-Gang/RTG-Sync



https://youtu.be/pk0j063cUAs

Summary

- Parallel real-time task scheduling
  - Hard to analyze on COTS multicore
  - Due to interference in shared memory hierarchy
- RT-Gang and RTG-Sync
  - **Analyzable** and **efficient** parallel real-time gang scheduling framework, implemented in Linux
  - Avoid and bound interference by design
  - Simple and achieves better analytic schedulability

KU THE UNIVERSITY OF KANSAS

Agenda

- Part 1. Time predictable software on COTS hardware

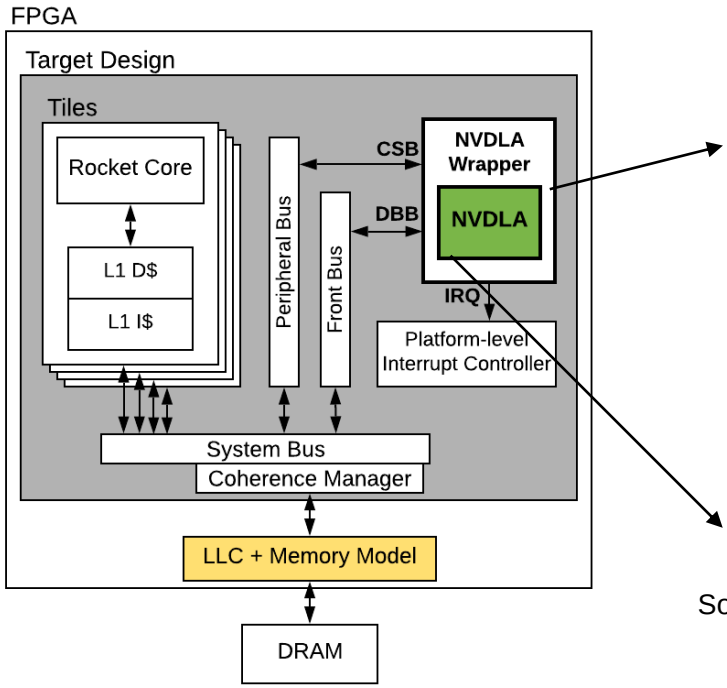
- **Part 2. Hardware/software collaboration for time predictability and security**

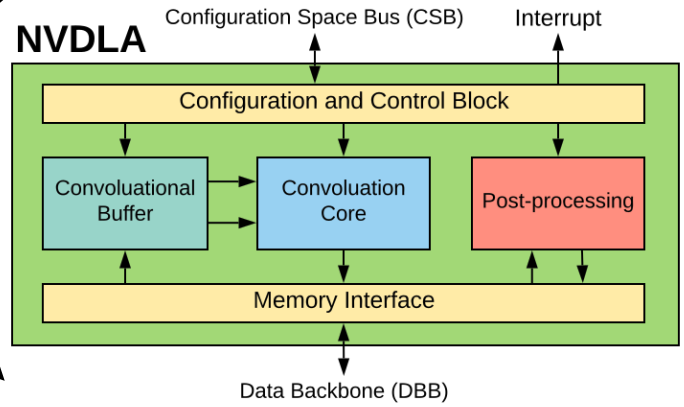
How to improve time predictability of high-performance processors?

[**EMC2'19**] Farzad Farshchi, Qijing Huang, and Heechul Yun. Integrating NVIDIA Deep Learning Accelerator (NVDLA) with RISC-V SoC on FireSim. *Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications*, 2019.

[**RTAS'20**] Farzad Farshchi, Qijing Huang, and Heechul Yun. BRU: Bandwidth Regulation Unit for Real-Time Multicore Processors. IEEE Intl. Conference on Real-Time and Embedded Technology and Applications Symposium, 2020. (to appear)

RISC-V SoC Testbed



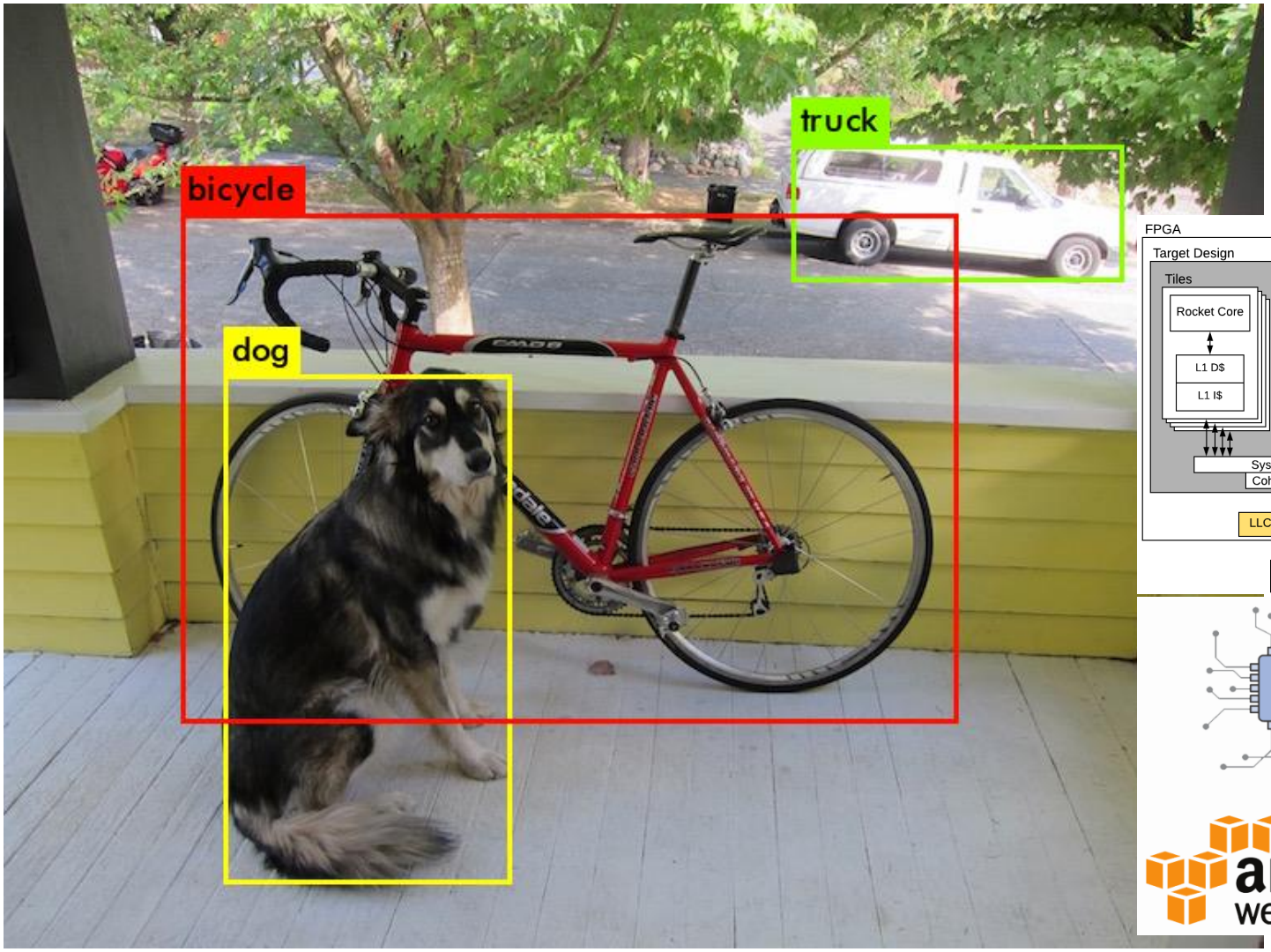


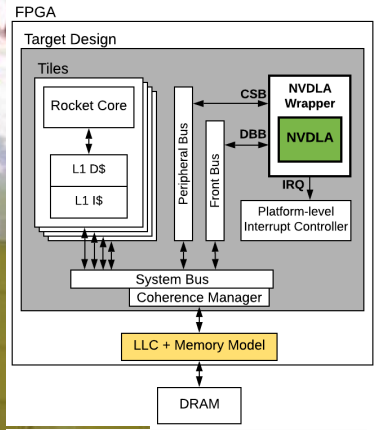
Source: NVIDIA, "The Nvidia Deep Learning Accelerator"

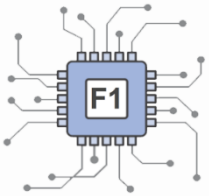
- Full-featured RISC-V cores (in-order/out-of-order) + hardware DNN accelerator on Amazon FPGA
  - Run Linux, YOLO v3 object detection

[**EMC2'19**] Farzad Farshchi, Qijing Huang, and Heechul Yun. Integrating NVIDIA Deep Learning Accelerator (NVDLA) with RISC-V SoC on FireSim. *Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications*, 2019.

YOLOv3 Object Detection

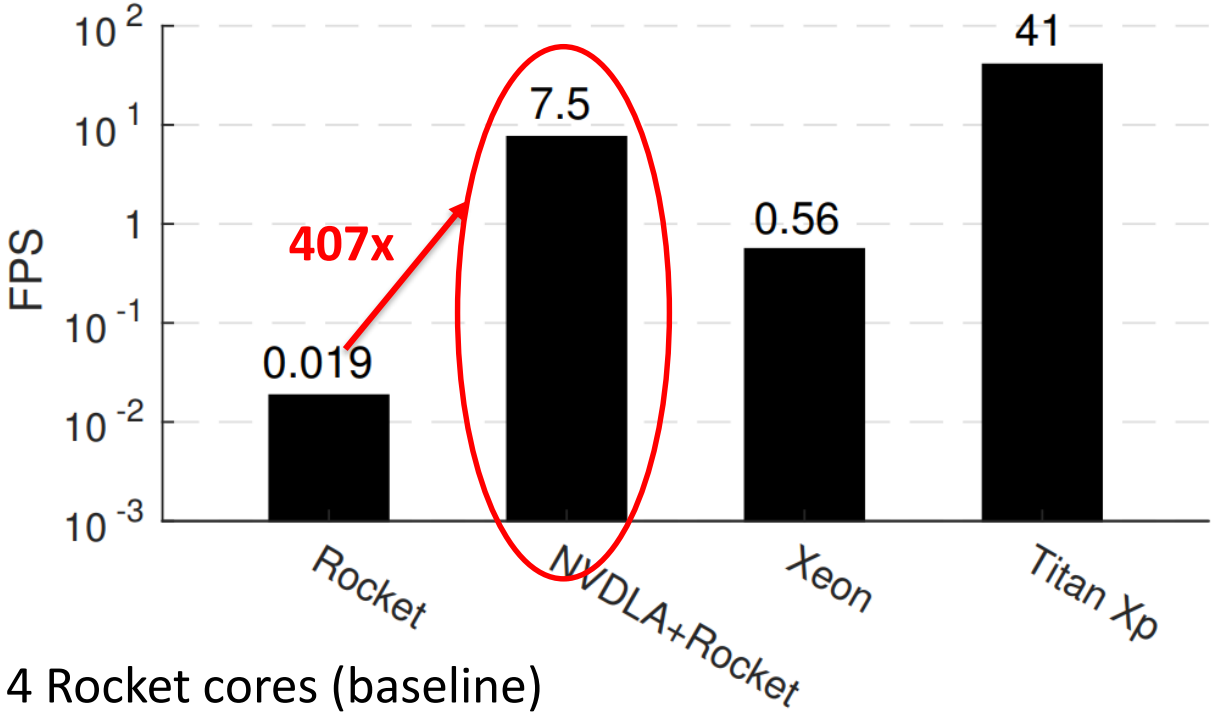






amazon web services

YOLOv3 Performance



- **Rocket**: 4 Rocket cores (baseline)
- **NVDLA+Rocket**: baseline + NVDLA
- **Xeon**: E5-2658 v3 (24 cores/48 threads)
- **Titan Xp**: Pascal arch, 3840 CUDA cores

KU THE UNIVERSITY OF KANSAS

https://devblogs.nvidia.com/nvdla/

NVIDIA. DEVELOPER

Join Login

NVIDIA Developer Blog

AI / DEEP LEARNING AUTONOMOUS MACHINES AUTONOMOUS VEHICLES DATA SCIENCE FEATURES GRAPHICS / SIMULATION HPC SMART CITIES

AI / DEEP LEARNING

NVDLA Deep Learning Inference Compiler is Now Open Source

By Rekha Mukund, Prashant Gaikwad and Mitch Harwell | September 11, 2019

Tags: compilers, Inference, Jetson, Machine Learning and AI, Object Detection

Designing new custom hardware ac
performance and efficiency with a n

Two years ago, NVIDIA opened the s
help advance the adoption of efficie
NVIDIA Jetson AGX Xavier Develope
open-source release of NVDLA's opt
point with the complete source for t

In this blog we'll explain the role tha
purpose-built hardware accelerator
NVDLA software and hardware desi

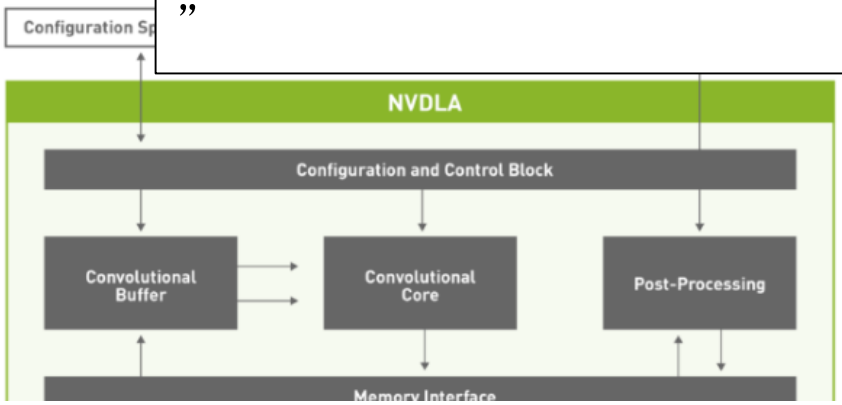
"

One of the best ways to get started is to dive right in with object detection using YOLOv3 on NVDLA with RISC-V and FireSim in the cloud.

…

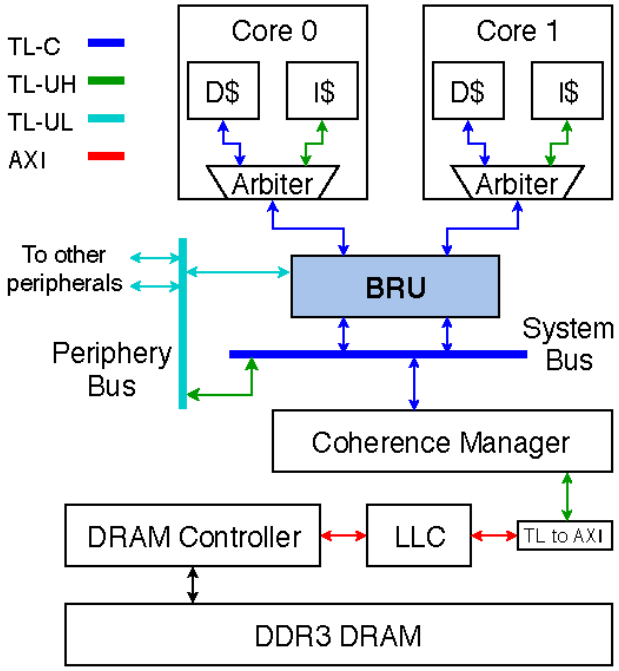
git clone https://github.com/CSL-KU/firesim-nvdla

"



KU THE UNIVERSITY OF KANSAS

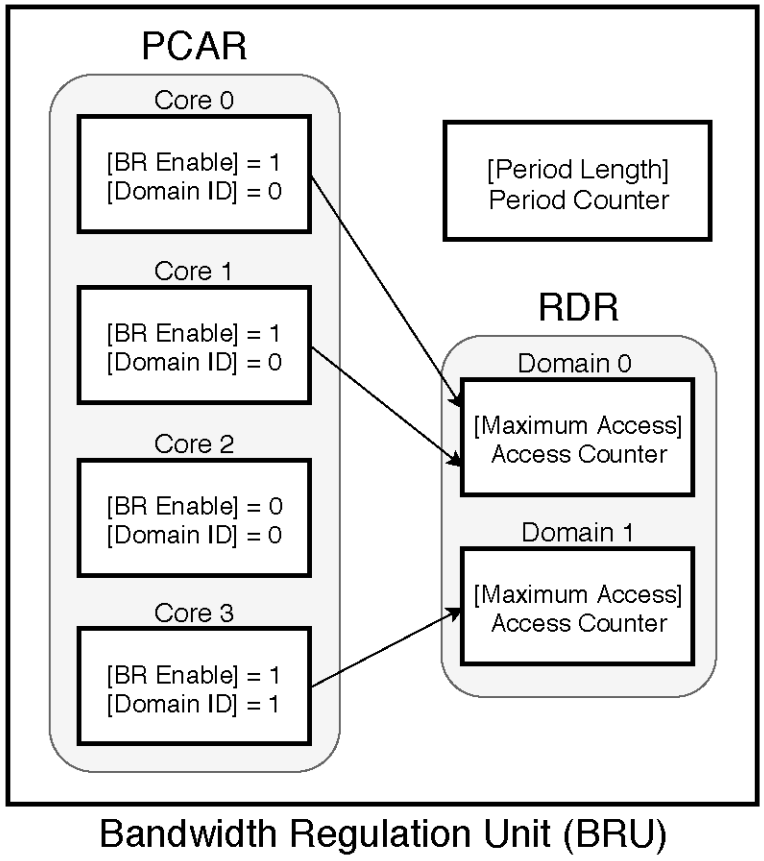
Bandwidth Regulation Unit (BRU)



- Regulate per-core/group memory bandwidth
- Drop-in addition to existing processor design

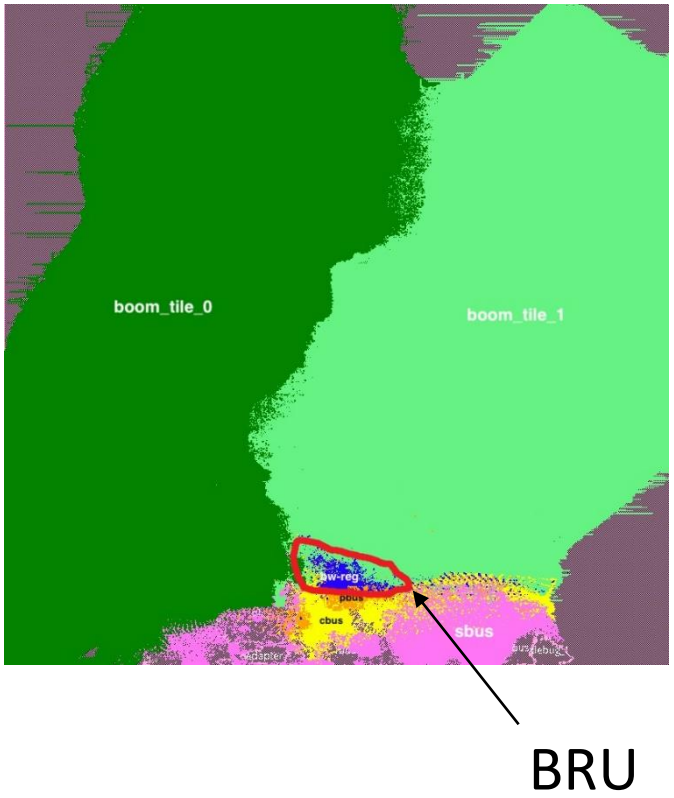
Bandwidth Regulation Unit (BRU)

- Access regulation
  - Regulate cache misses
- Writeback regulation
  - Regulate cache write-back
- Group regulation
  - Multiple cores can be regulated as a group



Dual-core BOOM with BRU

- BOOM: high-performance out-of-order RISC-V core
- Cadence synthesis result at 7nm node
- Less than 2% impact on max. frequency
- Less than 0.2% space overhead

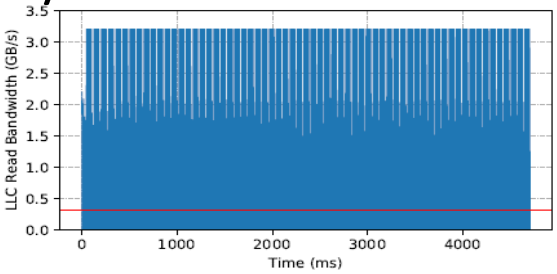


DUAL-CORE BOOM CHIP AREA BREAKDOWN

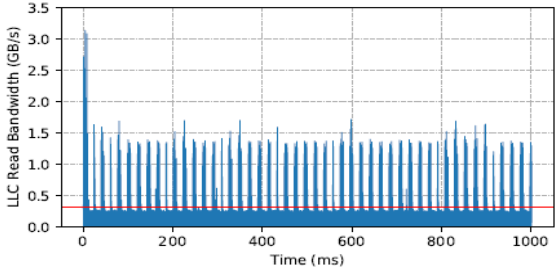
Modules	Area (μm^2)	Ratio
BRU	4,669	0.19%
Boom Core × 2	2,309,681	92.41%
Others (System Bus, Manager, etc.)	184,950	7.40%
Total	2,499,300	100%

Effects of BRU

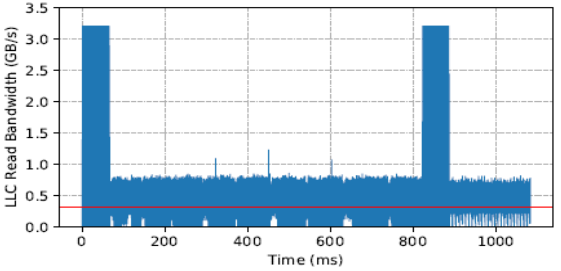
w/o BRU



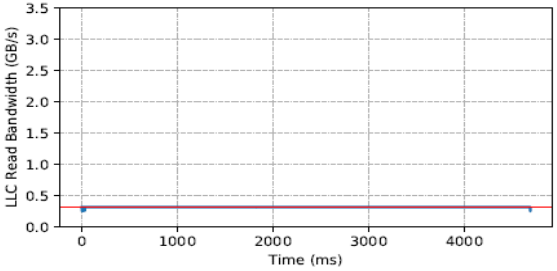
(a) disparity

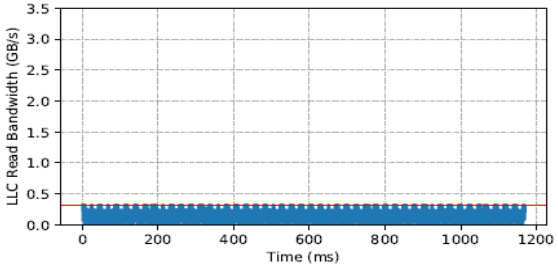


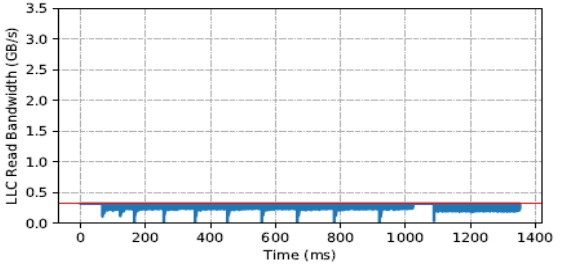
(b) localization



(c) svm







w/ BRU regulation (@320MB/s budget, 100ns period)

- BRU = MemGuard in hardware + alpha

Summary

- BRU
  - A synthesizable hardware IP that regulates memory traffic at the source (cores)
  - Demonstrates the feasibility of fast AND predictable processors
- Future work
  - Accelerator regulation support
  - More software/hardware co-design

How to improve security of high-performance processors?

[**DAC'19**] Jacob Michael Fustos, Farzad Farshchi, and Heechul Yun. SpectreGuard: An Efficient Data-centric Defense Mechanism against Spectre Attacks. Design Automation Conference, 2019

Speculative Execution Attacks

- Attacks exploiting microarchitectural **side-effects** of executing speculative (transient) instructions
- Recover secrets by measuring **timing differences** caused by the side-effects

Attack	Description
Variant 1 (Spectre) [16]	Bounds Check Bypass
Variant 1.1 [15]	Bounds Check Bypass Store
Variant 1.2 [15]	Read-only Protection Bypass
Variant 2 (Spectre) [16]	Branch Target Injection
Variant 3 (Meltdown) [18]	Supervisor Protection Bypass
Variant 3a [12]	System Register Bypass
Lazy FP [24]	FPU Register Bypass
Variant 4 [9]	Speculative Store Bypass
ret2spec [20]	Return Stack Buffer
L1 Terminal Fault [11, 26]	Virtual Translation Bypass

Known (subset of) speculative execution attacks

Spectre Attack (Variant 1)

```
if(x < array1_length){
    val = array1[x];
    tmp = array2[val*512];
}
........
```

- Assume `x` is under the attacker's control
- Attacker trains the branch predictor to predict the branch is in-bound

Spectre Attack (Variant 1)

```
if(x < array1_length){
    val = array1[x];
    tmp = array2[val*512];
}
.........
```

1. [ACCESS]

- Speculative execution of the first line **accesses** the secret (`val`)

Spectre Attack (Variant 1)

```
if(x < array1_length){
    val = array1[x];
    tmp = array2[val*512];
}
.........
```

2. [TRANSMIT]

- Speculative execution of the second, secret dependent load **transmits** the secret to a *microarchitectural state (e.g., cache)*

Spectre Attack (Variant 1)

```
if(x < array1_length){
    val = array1[x];
    tmp = array2[val*512];
}
........
```

3. [RECEIVE]

- Attacker **receives** the secret by measuring access timing differences (cache hit vs. miss) among the elements in the probe array
  - Various timing channels exist (e.g., cache)

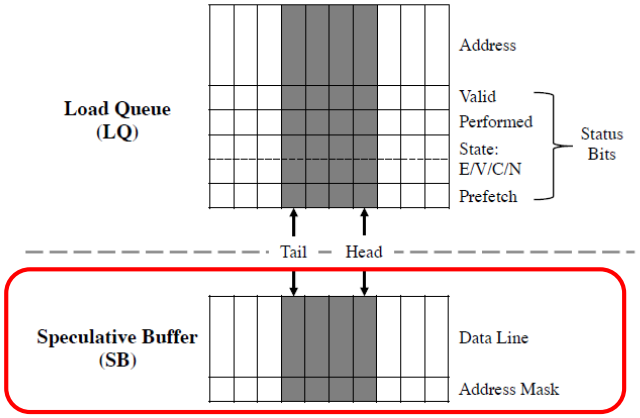
Existing Software Mitigation

```
if(x < array1_length){
      _mm_lfence();
    val = array1[x];
    tmp = array2[val*512];
}
```

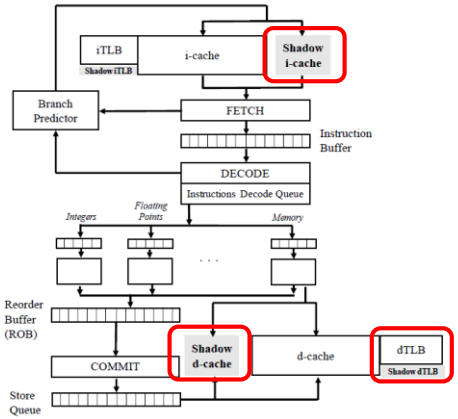
- Manually stop speculation
  - By inserting '`lfence`' instructions [Intel, 2018]
  - Or by introducing additional data dependencies [Carruth, 2018]
  - Error prone, high programming complexity, performance overhead

KU THE UNIVERSITY OF KANSAS

Existing Hardware Mitigation



InvisiSpec [Yan et al., MICRO'18]



SafeSpec [Khasawneh et al., DAC'19]

- Hide speculative execution
  - By buffering speculative results into additional "*shadow*" hardware structures
  - High complexity, high overhead (performance, space)

SpectreGuard

- Data-centric software/hardware co-design
  - Software tells hardware what **data** (not code) needs protection
  - Hardware selectively protects the identified **data** from Spectre attacks

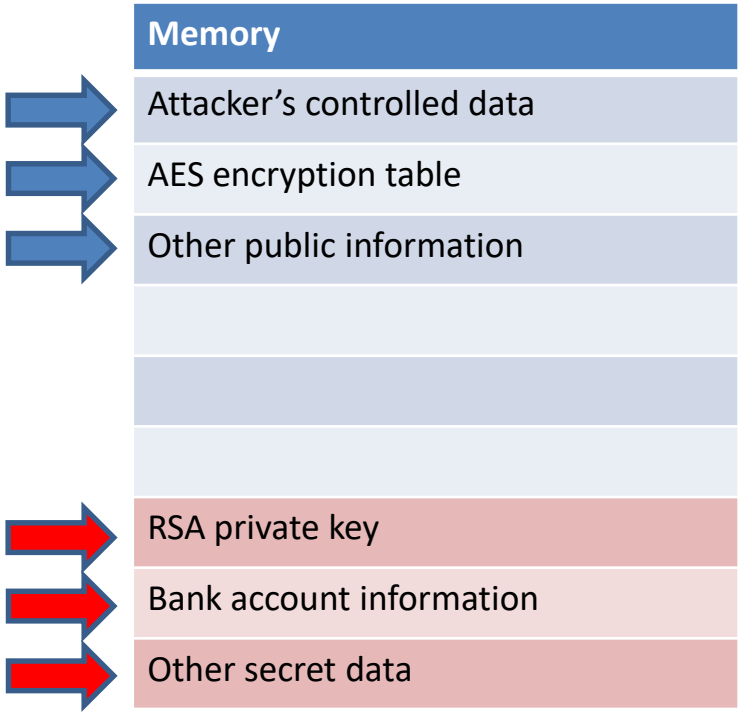
- Key observations
  - Not all data is secret
  - Not all speculative loads leak secret

KU THE UNIVERSITY OF KANSAS

[**DAC'19**] Jacob Michael Fustos, Farzad Farshchi, and Heechul Yun. SpectreGuard: An Efficient Data-centric Defense Mechanism against Spectre Attacks. Design Automation Conference, 2019

Obs. 1: Not All Data Is Secret

- Non-sensitive data
  - Most program code, data
  - **Optimize for performance**
- Sensitive (secret) data
  - Cryptographic keys, passwords, …
  - **Optimize for security**



Obs. 2: Not All Speculative Loads Leak Secret

```
if(x < array1_length){
    val = array1[x];
    tmp = array2[val*512];
}
.........
```

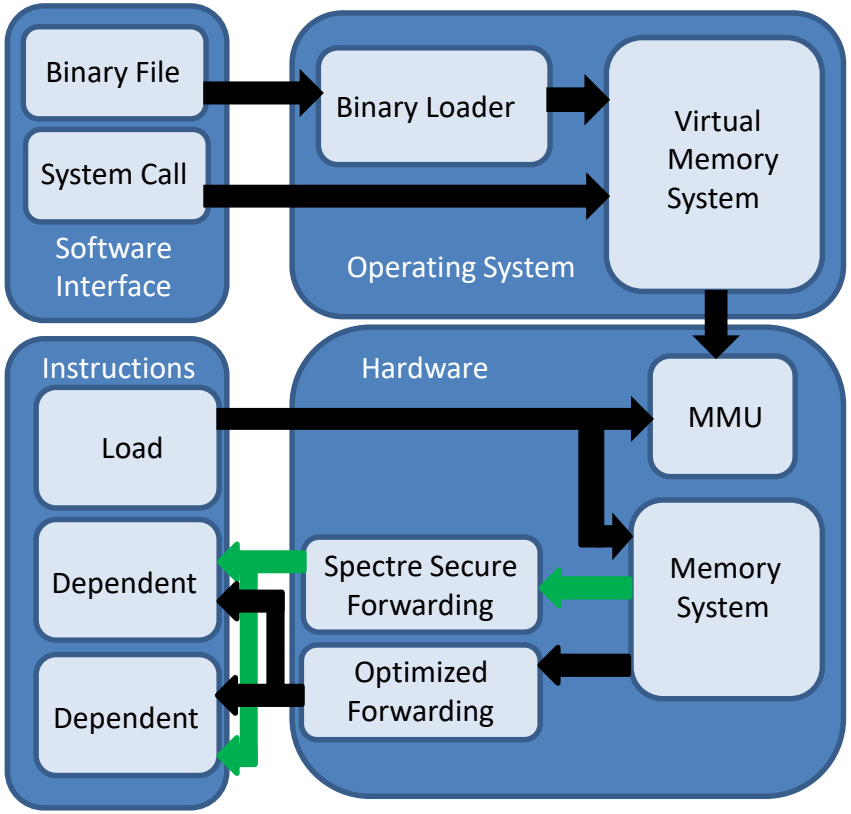
1. [ACCESS]
2. **[TRANSMIT]**

- The first load does **NOT** leak secret
- The second, **secret dependent load** leaks the secret
- Delay the secret dependent load until *after* the branch is resolved

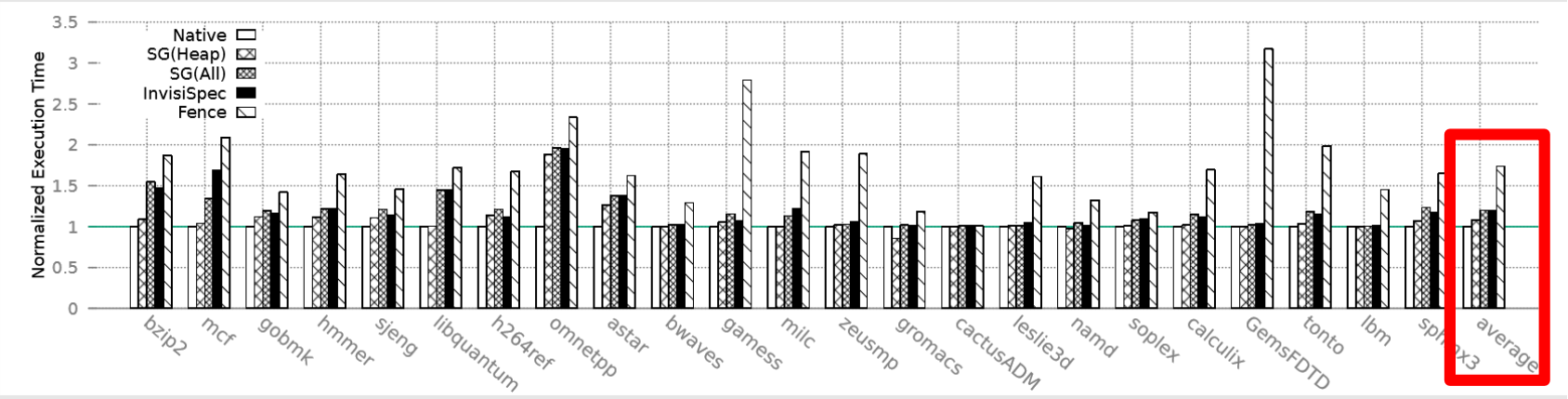
KU THE UNIVERSITY OF KANSAS

SpectreGuard Approach

- Step 1: Software tells OS what data is secret
- Step 2: OS updates the *page table* entries
- Step 3: Load of the secret data is identified by MMU
- Step 4: secret data forwarding is **delayed** until safe



Results of SPEC2006 Benchmarks



- Good protection at low **controllable** overhead
- SpectreGuard enables targeted security and performance trade-offs

KU THE UNIVERSITY OF KANSAS

Summary

- Speculative execution attacks
  - Affect all high-performance out-of-order processors
  - Existing software mitigation suffers high programming complexity/overhead
  - Hardware only mitigation is costly

- SpectreGuard
  - A data-centric software/hardware collaborative defense mechanism
  - Low programming effort (identifying secret **data**, not vulnerable code)
  - Low hardware cost (no additional "shadow" structure)
  - Effective, targeted defense against Spectre attacks

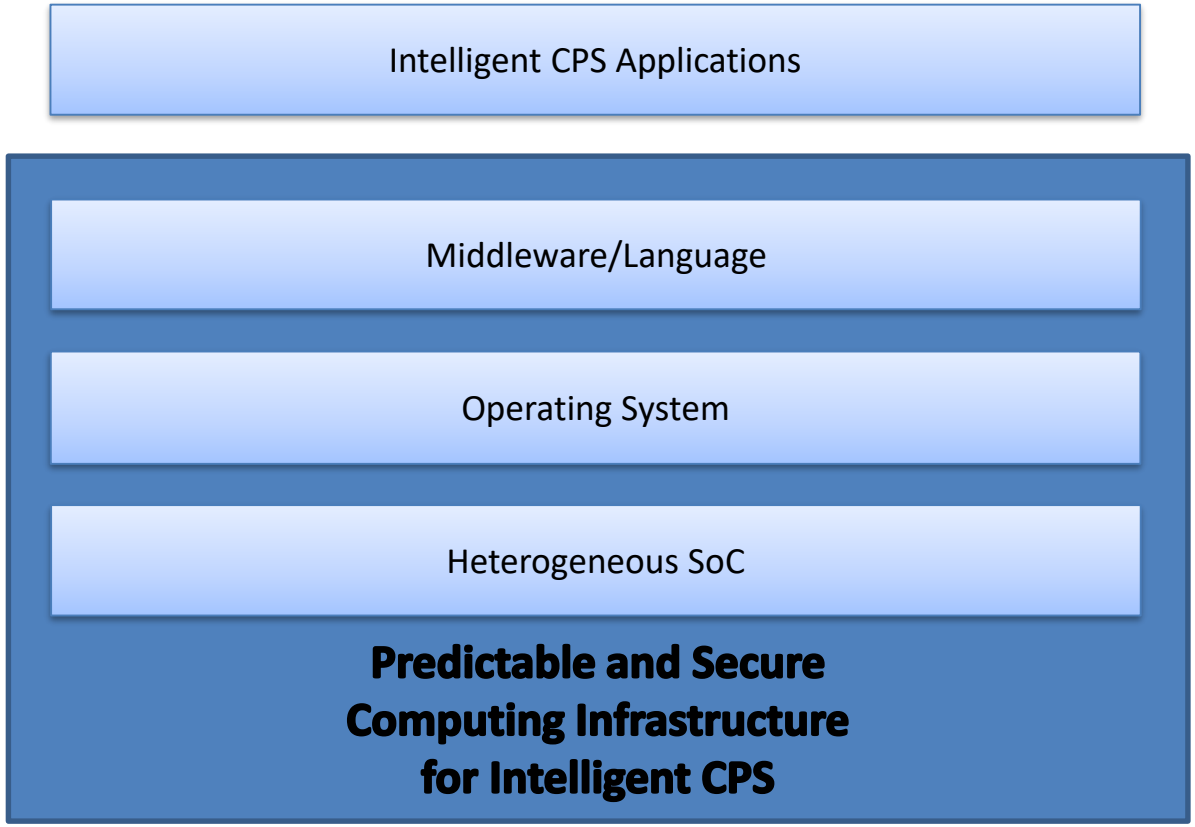
https://github.com/CSL-KU/SpectreGuard

KU THE UNIVERSITY OF KANSAS

Conclusion

- Smart scheduling and OS support can provide time predictability on COTS hardware
- Small changes in COTS hardware can provide both time predictability and high performance
- Our research develops fundamental computing infrastructure technologies to enable predictable and secure computing for intelligent CPS

Our Vision



- Holistic, cross-layer: from chips to applications

KU THE UNIVERSITY OF KANSAS

Recent Publications

1. [**RTAS'20**] Farzad Farshchi, Qijing Huang, and Heechul Yun. BRU: Bandwidth Regulation Unit for Real-Time Multicore Processors. *IEEE Intl. Conference on Real-Time and Embedded Technology and Applications Symposium*, 2020. (to appear)
2. [**DAC'19**] Jacob Michael Fustos, Farzad Farshchi, and Heechul Yun. SpectreGuard: An Efficient Data-centric Defense Mechanism against Spectre Attacks. *Design Automation Conference*, 2019
3. [**ECRTS'19**] Renato Mancuso, Heechul Yun, Isabelle Puaut. Impact of DM-LRU on WCET: a Static Analysis Approach. *Euromicro Conference on Real-Time Systems*, 2019
4. [**RTAS'19-2**] Waqar Ali and Heechul Yun. RT-Gang: Real-Time Gang Scheduling Framework for Safety-Critical Systems. *IEEE Intl. Conference on Real-Time and Embedded Technology and Applications Symposium*, 2019.
5. [**RTAS'19-1**] Michael Garrett Bechtel and Heechul Yun. Denial-of-Service Attacks on Shared Cache in Multicore: Analysis and Prevention. *IEEE Intl. Conference on Real-Time and Embedded Technology and Applications Symposium*, 2019 **Outstanding Paper Award**
6. [**EMC2'19**] Farzad Farshchi, Qijing Huang, and Heechul Yun. Integrating NVIDIA Deep Learning Accelerator (NVDLA) with RISC-V SoC on FireSim. *Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications*, 2019.
7. [**RTCSA'18**] Michael Garrett Bechtel, Elise McEllhiney, Minje Kim, Heechul Yun. DeepPicar: A Low-cost Deep Neural Network-based Autonomous Car. *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications,* 2018
8. [**ECRTS'18**] Waqar Ali, Heechul Yun. Protecting Real-Time GPU Applications on Integrated CPU-GPU SoC Platforms. *Euromicro Conference on Real-Time Systems*, 2018
9. [**ECRTS'18**] Farzad Farshchi, Prathap Kumar Valsan, Renato Mancuso, Heechul Yun. Deterministic Memory Abstraction and Supporting Multicore System Architecture. *Euromicro Conference on Real-Time Systems*, 2018
10. [**RTSJ'17**] Prathap Kumar Valsan, Heechul Yun, Farzad Farshchi. Addressing Isolation Challenges of Non-blocking Caches for Multicore Real-Time Systems. *Real-time Systems*, Vol: 53, Issue: 5, pp: 673–708, 2017
11. [**TC'17**] Heechul Yun, Waqar Ali, Santosh Gondi, Siddhartha Biswas. BWLOCK: A Dynamic Memory Access Control Framework for Soft Real-Time Applications on Multicore Platforms. *IEEE Transactions on Computers*, Vol: 66, Issue: 7, pp: 1247-1252, 2017
12. [**RTCSA'16**] Prasanth Vivekanandan, Gonzalo Garcia, Heechul Yun, Shawn Keshmiri. A Simplex Architecture for Intelligent and Safe Unmanned Aerial Vehicles. *IEEE Intl. Conf. on Embedded and Real-Time Computing Systems and Applications*, 2016. **Best Student Paper Nomination**
13. [**RTAS'16**] Prathap Kumar Valsan, Heechul Yun, Farzad Farshchi . Taming Non-blocking Caches to Improve Isolation in Multicore Real-Time Systems. In *IEEE Intl. Conference on Real-Time and Embedded Technology and Applications Symposium*, 2016. **Best Paper Award**
14. [**TC'16**] Heechul Yun, Gang Yao, Rodolfo Pellizzoni, Marco Caccamo, and Lui Sha. Memory Bandwidth Management for Efficient Performance Isolation in Multi-core Platforms, *IEEE Transactions on Computers,* Vol 65, Issue 2, 2016, pp. 562 – 576. **Editor's Pick of the year 2016**

KU THE UNIVERSITY OF KANSAS

Full List: http://www.ittc.ku.edu/~heechul/pub.html

Thank You!

Acknowledgement:

This research has been supported by NSA Science of Security initiative contract #H98230-18-D-0009 and NSF CNS 1718880, 1815959.

KU THE UNIVERSITY OF KANSAS